Robot Framework
自动化测试修炼宝典

齐 涛 著

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书分为 3 个部分进行介绍。在第 1 部分筑基篇中，主要有自动化测试概述、Robot Framework 两章的内容；在第 2 部分小乘篇中，主要有 Web 自动化测试、C/S 自动化测试、数据库自动化测试、接口自动化测试、RF 内置测试库、持续集成自动化测试、移动自动化测试总共七章的内容；在第 3 部分大乘篇中，主要有自定义你的 RF 一章的内容。全书的自动化测试主要指功能自动化测试。

本书适合阅读的人群很广，基本上想做自动化测试的人都可以阅读。本书希望用更多的实例来引导读者上手，刚开始学习的读者跟着做会比较好。本书涵盖了在自动化测试中遇到的大部分场景，不但内容充实、逻辑严密，且图文并茂、语言生动。对 Robot Framework 使用者来说，本书更是一部人人必备在案头的工具书。

图书在版编目（CIP）数据

Robot Framework 自动化测试修炼宝典 / 齐涛著. —北京：电子工业出版社，2015.11
ISBN 978-7-121-27405-3

Ⅰ. ①R… Ⅱ. ①齐… Ⅲ. ①软件工具－测试 Ⅳ. ①TP311.56

中国版本图书馆 CIP 数据核字（2015）第 245171 号

策划编辑：董　英
责任编辑：徐津平
印　　刷：北京京师印务有限公司
装　　订：北京京师印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　　印张：18.5　　字数：371 千字
版　　次：2015 年 11 月第 1 版
印　　次：2015 年 11 月第 1 次印刷
印　　数：4000 册　　定价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

推荐序一

我们已经进入了移动互联网的时代，企业在市场、渠道、产品、服务各方面都面临着新的挑战，每个成功企业都在培养快速适应变化的能力，这就是我们时常说的企业敏捷性的重要组成因素。对于企业的 IT 部门来说，面对着愈发不确定的客户需求，快速并高质量地完成开发工作，使需求早日上线，从而能够尽早收集市场反馈，优化产品或服务，是必须着重解决的问题。在追求快速上线的同时，质量底线是我们必须坚守的红线，在此过程中，自动化回归测试技术是一种行之有效的保障手段。本书介绍的 Robot Framework 框架就是非常优秀的自动化回归测试框架，值得从事开发测试的同行仔细研究。

从 2011 年起，我以咨询顾问的身份进入平安科技，帮助其进行敏捷转型，在那时认识了本书作者。在一些试点项目中，我们要求开发团队在提升交付速度的同时保证质量，但原有的 QTP 工具不能满足要求。经过调研，我们选定了“Robot Framework + Selenium”的开源技术体系来替代 QTP，完成界面自动化回归测试工作，并在实践中取得了非常良好的效果，得到了各方的肯定和赞誉。后来，我们用 Robot Framework 结合 Requests 来进行 HTTP 接口的自动化回归测试，进一步夯实了分层自动化测试体系，为平安科技日后的全面敏捷转型打下了坚实的基础。

在这一过程中，齐涛潜心钻研，经过不懈努力逐步承担起公司内部 Robot Framework 的支持和推广工作，自己也成为了这方面的专家。本书是他的开山之作，具备很高的实操性，相信读者可以从本书中学到诸多来自一线的实践经验。最后，祝大家阅读愉快！

Agilean 咨询公司创始人、平安科技敏捷转型总顾问　吴穹

推荐序二

移动互联网发展到 2015 年，算是进入了一个白热化阶段，开发和测试的技术发展都已经突破了好几次瓶颈，但现在依然在追求更高的技术和产品质量。而软件测试人员算是在这个热潮中比较辛苦的，正巧软件测试在国内也处在快速发展和变化期，所以无论是测试技术，还是本书中提到的测试分层，都是对测试人员极大的挑战。

我认识道长（本书作者）也有好多年了，道长为人非常谦逊，做事情比较踏实，不像我那么高调，同时对技术也有很大的追求。我依旧秉持我做事的原则，就如同我要去测试一个产品，先要对这个产品的业务和技术架构有深入了解一样，我是通读了本书之后才开始写序的。我能体会到道长写这本书的不容易，就如作者在书的最后提到的，Robot Framework 本身在国内受众不大，所以道长写这本书在我看来一方面是给用 Robot Framework 的同行一些自己的经验，另一方面真的就是自己对自己的一个总结。我在这里呼吁大家真要尊重写书的人（当然那种纯理论忽悠的不包括在内），因为一本书的背后都有不为人知的很多心酸，没有一些压力和自己的坚持是无法完成的。一本书的出版对于作者而言就如同十月怀胎，如同自己的孩子一样亲切。

我还是回过来说这本书吧。本书就如作者自身定义的，更像一本操作手册，虽然说的是 Robot Framework 这样一个工具，但其实涉及的面非常广。其中包括但不仅限于 Web 自动化测试、接口自动化测试，以及现在大家很关心的移动无线的自动化测试，书的最后一部分也提到了使用 Robot Framework 去做持续集成，以及如何结合移动最新的框架 Appium 来做自动化测试。移动互联网测试行业还是有蛮多人认识我的，也知道我的风格。首先这肯定不是一本“忽悠”的书，相反，这本书在每个章节都有大量的实践，以及需要每个读者去操作了才能够真正理解的内容。道长也非常贴心地在书中提供了知识点和案例，写明了官方的下载地址以及自己的 Github 地址，让大家能够更好地开展学习。不过我在这里依然需要提醒读者，移动互联网的技术和知识迭代都是非常快的，而且技术栈会比较深，所

以大家在读相关书籍的时候要自己学会 Google 相关技术的官方文档，从而有一个全面的理解，千万不要指望在书中找到所有的答案，在移动互联网时代这是不现实的。

Robot Framework 我觉得还是很有必要学习一下的，很多测试行业的同行其实会比较疑惑在项目中的自动化用例怎么大批量编写、管理等问题，我个人觉得 Robot Framework 就是一个不错的粘合剂。Robot Framework 能够结合各种测试工具进行测试，同时也能够更方便和有条理地管理我们的测试用例。Robot Framework 本身在国内的中文资料很少，本书对于初学者来讲会有非常大的帮助，同时也让用过 Robot Framework 的同行们能够对 Robot Framework 有一个更全面的了解。

在互联网行业做一名测试工程师很难，在移动互联网行业做一名测试工程师更难。移动互联网行业的测试工程师需要面临非常多的测试工具、框架和技术，不得不说在这其中，Robot Framework 是个不错的工具，相信它能够帮助大家在项目中将测试技术更好地落地。

最后要再次感谢道长对于本书的付出，也预祝本书大卖，并且能够帮助到更多的人。

写于 2015 年

蚂蚁金服资深测试开发工程师、Testerhome 测试技术社区创始人之一、《大话移动 App 测试》作者　陈晔（Monkey）

前　言

写这本书之前一定要先感谢一个人——吴穹博士，如果不是吴穹博士来平安科技做咨询顾问，向我们推荐了 Robot Framework（下面简称 RF）这个框架，我想或许我还在玩 QTP 呢，又或者用刘兴罡的 PLSA（这也是一个很棒的工具）。吴穹博士是我的引路人，他也是很多目前在用 RF 做自动化测试的同行们的引路人，非常感谢吴穹博士。

我不是“科班”出身，也许大家都想不到，我大学学习的专业是市场营销，可惜专业课的知识都还给老师了，凭着对 IT 的喜爱，经历了种种磨难，成功“混入”了 IT 圈，还做了测试。测试其实是一个“高危职业”，要比产品经理（或 SA 需求分析）想得全面，要比开发懂需求，要有很强的发散思维，要能读懂代码，必要的时候还要会写个代码。说点做测试的同行不爱听但却是事实的话，很多人是因为觉得做测试比较简单才入的这行。说测试是一个“高危职业”，那是因为当生产上出了问题，90%的人（领导）都会问：“测试的时候为什么没有发现？”（这是好的语气，不好的语气就是：“你们是怎么测试的？这都没发现？”）也不知道我怎么当初就头脑发热地选择了做测试，也不知道为什么当初在华为做外包的时候，天天加班还觉得很充实。在这里我奉劝各位，测试有风险，入行需谨慎！

我在 2008 年进入平安科技时，是以外包的身份进来的，进来之前“摸”了一个星期的 QTP，进来以后就专职负责做 QTP 自动化测试案例。当时的组里本身有一批 QTP 案例，都是录制出来的，那时候最痛苦的是由需求变更导致的修改案例，每次发版本要跑回归，有时候要持续到凌晨两三点，确保所有的自动化测试案例都没问题了，才能封版上线。后来逐步改善案例，逐步抛弃掉了录制回访，从描述式编程到后来用的 Framework Manager 框架，基本上可以说把 QTP“玩”到极致了吧，其实到现在我偶尔还在用 QTP，比如可以用来帮我批量打开十几个远程桌面什么的。

后来经过朋友介绍进入了平安科技，从外包转为了内部员工。那时候新进入的组里基

本上没有什么自动化测试案例，大部分时间其实也是在手工测试。到了 2011 年 9 月左右，借着组内的一个项目，吴穹博士作为咨询顾问，推荐 Robot Framework 给我们。当我学习了这个工具后，被它深深吸引住了，对于我这样用惯了 QTP 的懒人来说，RF 这种轻量又全面的工具，基本上完胜 QTP。也是差不多从那个时候开始，我在博客上断断续续地写起了 RF 的相关博客，也相当于学习笔记吧。同时我也顺便学起了 Python，有需要的时候就会看 RF 的底层源代码，到后来可以适当地改动一些代码来扩展 RF 库，写代码对我这样的“门外汉”来说还是比较困难的，改代码就是我比较擅长的了，最不济还可以“照猫画虎”。

平安科技的自动化测试工具最早的时候基本上只有 QTP，后来增加了 Robot Framework 和刘兴羿自己开发的 PLSA，变成了“三足鼎立”。然后 QTP 直接被废弃，变成“双雄争霸”了。其实我倒是觉得没什么争的，RF 和 PLSA 各有自己的特点，RF 降低了大家做自动化测试的门槛，人人都可以做自动化测试，我觉得这样挺好的。从另一个角度刺激了专职的测试人员要提升自身的竞争力。而 PLSA 还是需要有些 Java 开发经验的，当然它也可以零基础入门，但是我觉得用 Eclipse 写自动化脚本还是太重了。顺带提一句，刘兴羿后来做了 RF 的 PLSA 测试库，也就是在 RF 上也可以用 PLSA，这算是共赢吧。

经历了这些年的自动化测试，其实最头疼的是领导的思维，被那些“大忽悠”给“忽悠”了。通常大佬们最容易被“忽悠”的有三点：第一点，自动化率。盲目地追求 100% 的自动化率，每年都把提高自动化率作为 KPI 之一；第二点，发现缺陷数。自动化测试案例能发现多少缺陷，自动化案例运行失败不一定是缺陷导致的，可能是执行的环境有问题、需求变更以及其他的一些情况。真正在失败案例中能发现缺陷的，可能都不会超过 10%；第三点，节省人力。认为多做自动化测试案例能节省人力，怎么说呢，案例越多其实反而不节省人力，除非你的系统一年没多少需求变更。

遇到这样的领导你说怎么办？说服领导？换个公司？想办法轻松地完成任务？对我这样的懒人来说，我会先想办法轻松地完成任务，顺便提升一下技术实力。因为我一直坚信，懒人的存在促使科技进步，所以 Robot Framework 正好合适，初级入门可以完成任务，深入研究可以提升技术。

本书适合阅读的人群很广，基本上想做自动化测试的人都可以阅读。本书的自动化测试主要指功能自动化测试。适合初学者入门学习。对于最深入的进阶部分，我只会简单提一下，因为我觉得到了那个层次，大家已经可以自由发挥了，所以可能不太适合“老鸟们”了，当然入手收藏也是不错的。我不太擅长概念性的内容，所以涉及概念性的内容可能会比较少，或者很快带过。我希望用更多的实例来引导大家上手，刚开始学习的读者跟着做

会比较好，书里面的案例我也会与大家分享。

本书第 1 章是一些自动化测试的概念性内容，并没有深入探讨；第 2 章是整个 Robot Framework 的基础，建议初学者多花点时间学习；从第 3 章到第 9 章分别介绍了不同类型的自动化测试、持续集成和移动自动化测试；最后第 10 章作为深入学习的部分，内容不是很多，主要是为了给大家一个入门的方向，具体还要看大家各自发挥。

特别鸣谢本书的审稿人陈晔和徐毅，感谢两位的辛苦付出，提出了不少修改意见。感谢百忙之中帮忙进行封面设计的陈争，也感谢出版社的各位编辑帮忙进行排版设计。

齐涛

目　　录

第一部分　筑基篇

第 1 章　自动化测试概述 2

1.1　分层自动化测试 2
1.2　界面自动化测试工具 3
1.3　自动化测试做到什么样算好 4
1.4　小结 5

第 2 章　Robot Framework 6

2.1　框架介绍 6
2.2　安装指南 8
2.2.1　Windows 平台 8
2.2.2　Mac 平台（Linux 平台可参考） 15
2.2.3　pip 安装 19
2.3　第一个案例 19
2.3.1　打开 RIDE 20
2.3.2　创建工程和测试套 21
2.3.3　创建案例 23
2.3.4　写一行脚本 24
2.3.5　运行案例 25
2.3.6　查看运行结果 26
2.3.7　小结 28
2.4　RIDE 工具 28

2.4.1 工具介绍 28
2.4.2 菜单栏&工具栏 29
2.4.3 工作区 45
2.5 工程、测试套件、测试案例 49
2.5.1 Project 工程 49
2.5.2 Test Suite 测试套件 53
2.5.3 Test Case 测试案例 56
2.5.4 三者关系 58
2.6 测试库 59
2.7 Resource 资源文件 65
2.7.1 新建资源 65
2.7.2 快捷菜单 66
2.7.3 Settings 设置项 66
2.7.4 加载资源 67
2.7.5 External Resources 外部资源 68
2.8 变量和常量 71
2.8.1 变量与常量基础 72
2.8.2 Scalar 变量 79
2.8.3 List 变量 88
2.8.4 变量转换 97
2.8.5 其他变量 104
2.8.6 变量文件 104
2.9 Keyword 关键字 105
2.9.1 用户关键字 105
2.9.2 传入参数 Arguments 108
2.9.3 Return Value 返回值 116
2.10 循环&分支 127
2.10.1 循环 127
2.10.2 分支 134
2.10.3 二者结合 142
2.11 运行界面 143
2.12 小结 149

第二部分　小乘篇

第 3 章　Web 自动化测试 152

3.1　Selenium 152
3.2　Selenium2Library 安装 153
3.3　Selenium2Library 常用关键字 155
3.3.1　browserManagement 155
3.3.2　Cookie 157
3.3.3　Elements 157
3.3.4　JavaScript 160
3.3.5　screenshot 160
3.3.6　waiting 160
3.4　测试案例设计 161
3.4.1　案例设计 Step1 162
3.4.2　案例设计 Step2 164
3.4.3　案例设计 Step3 166
3.5　测试案例 demo 168
3.5.1　使用 demo 前的准备 168
3.5.2　Selenium2Library-demo 170
3.6　常见问题 177
3.7　小结 178

第 4 章　C/S 自动化测试 179

4.1　AutoIt 179
4.2　AutoItLibrary 安装 180
4.3　AutoItLibrary 关键字 181
4.4　测试案例 182
4.4.1　计算器案例 182
4.4.2　结合 Selenium2Library 处理对话框 183
4.4.3　结合 Selenium2Library 处理上传下载 185
4.5　小结 189

第 5 章　数据库自动化测试 190

5.1　数据库测试介绍 190
5.2　DatabaseLibrary 和 cx_Oracle 安装 191
5.3　DatabaseLibrary 关键字 191
5.4　测试案例 191
5.4.1　Oracle 数据库 191
5.4.2　sqlite3 数据库 193
5.5　常见问题 194
5.6　小结 195

第 6 章　接口自动化测试 196

6.1　接口测试 196
6.2　requestsLibrary、requests 安装 197
6.3　requestsLibrary 关键字 197
6.4　测试案例 197
6.5　小结 201

第 7 章　RF 内置测试库 202

7.1　测试库介绍 202
7.2　BuiltIn 203
7.2.1　Convert 203
7.2.2　Verify 204
7.2.3　Variables 205
7.2.4　RunKeyword 206
7.2.5　Control 207
7.2.6　Misc 208
7.2.7　强大的 Evaluate 208
7.3　String 212
7.3.1　Convert 213
7.3.2　Line 213
7.3.3　String 214
7.4　Collections 215

7.4.1 List 215
7.4.2 Dictionary 215
7.5 OperatingSystem 216
7.5.1 Env 216
7.5.2 File 217
7.5.3 Directory 217
7.5.4 Path 218
7.6 Process 218
7.7 XML 219
7.8 其他测试库 220
7.9 小结 220

第 8 章　持续集成自动化测试 221

8.1 Jenkins 安装与配置 221
8.1.1 Jenkins 简介 221
8.1.2 安装 Jenkins 222
8.1.3 配置节点 223
8.1.4 安装插件 228
8.2 Jenkins 上执行 RF 自动化测试 230
8.2.1 创建 Job 231
8.2.2 配置 Job 232
8.2.3 控制 Job 239
8.2.4 RF 插件作用 240
8.2.5 多配置 Job 241
8.3 小结 244

第 9 章　移动自动化测试 245

9.1 Appium 介绍 245
9.2 Android 自动化测试 250
9.2.1 模拟器安装 250
9.2.2 测试案例 253
9.2.3 Android 对象识别 256

9.3 iOS 自动化测试 257
9.3.1 测试案例 257
9.3.2 iOS 对象识别 260
9.4 小结 263

第三部分 大乘篇

第 10 章 自定义你的 RF 266
10.1 修改 Selenium2Library 测试库 266
10.2 编写测试库 270
10.2.1 测试库分类 271
10.2.2 测试库结构 271
10.2.3 测试库命名 273
10.2.4 测试库头部 273
10.2.5 测试库配置项 274
10.2.6 测试库文档 274
10.2.7 测试库关键字命名 275
10.2.8 测试库关键字参数 275
10.2.9 测试库的参数 276
10.2.10 测试库关键字返回值 276
10.2.11 测试库日志打印 276
10.2.12 对测试库做测试 277
10.2.13 发布测试库 277
10.3 小结 279

结语 280

参考资料 282

第一部分　筑基篇

- 第 1 章　自动化测试概述
- 第 2 章　Robot Framework

1

第 1 章
自动化测试概述

本章主要介绍自动化测试的分层、常见自动化测试工具，以及关于自动化测试的一点想法。

1.1 分层自动化测试

图 1-1-1 是经典的测试金字塔，出自 Martin Fowler 的博客（http://martinfowler.com/bliki/TestPyramid.html）。用它来形容目前测试投入的价值是比较合适的，同样也可用来说明自动化测试的投入价值。

- UI 层：界面自动化测。可以看出它的价值最小，所以适当的界面自动化测试是有必要的，但是没必要 100%都自动化。
- Service 层：接口自动化测试。它的价值居中，覆盖大多数主要的接口是比较合适的。
- Unit 层：单元测试。最有价值的测试，不过我个人认为这是开发人员应该做的事

情。如果让测试人员做也不是不可以，只是对测试人员的要求高了一些，而且最好和开发结对编程才能做好。我期望开发人员都用 TDD。

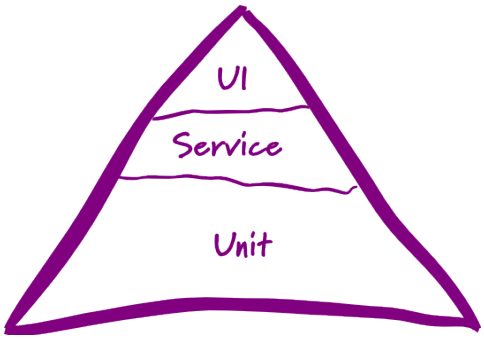

图 1-1-1

本书中的自动化测试基本上都是指前两者，特别是 UI 自动化测试，虽然价值小，但是对测试人员来说是比较容易提升自身实力的一技之长。什么？你说让测试人员去学编程做单元测试，对于大部分测试人员，特别是像我这样的非“科班”出身的测试人员，还是比较有难度的。我见过很多测试人员，也面试过很多测试外包，这里面计算机专业毕业的比较少，其他专业的倒是很多，学计算机专业的再会点编程的就更少了。

希望测试金字塔能稍微让大家有个认识，知道自己做的东西的价值。

1.2　界面自动化测试工具

既然说到了界面自动化测试，即 UI 自动化测试，比较常见的工具有：QTP、AutoIt、Selenium 等。像 QTP 经历了很多版本，最新的版本好像叫 UFT 了。对初学者来说，录制回放是相当容易上手的，除了录制，QTP 主要用 VBScript 脚本编写代码。由于接触了 VBS，后来还找到了一个专门用 VBS 做 Web 测试的工具，叫 AutonomyV；AutoIt 主要是 Windows 下的 UI 测试，我要加一句是标准的 Win32 对象，一些非标准的，比如 Delphi 或者 PowerBuilder 之类做出来的对象，AutoIt 就“无能为力”了；Selenium 其实算是做 Web 测试最全面的工具了，支持的浏览器种类全面，而且又是直接在浏览器中操作，接近真实操作。

其实后面这两个工具，现在在 RF 里都有对应的测试库，可以用来进行自动化测试，在后面的章节会介绍。而对于 QTP 来说，RF 基本上已经替代了它的功能，唯一欠缺的就是录制了。

简单对比一下 QTP 和 RF。这是吴穹博士在推荐我们放弃 QTP、采用 Robot Framework 的时候做的一个比较表，表内每项满分为 5 分，分数范围 1～5。总体来说，RF 方案是占优的，见表 1-2-1。

表 1-2-1

对比项	QTP方案	RF方案
忠实模拟用户行为	4.5	3.5
浏览器支持	3	3.5
动态行为支持	3	4.5
测试速度	2.5	3
支持数据驱动的测试	3	4.5
中文支持	5	4.5
测试用例稳定性	4	4
编写用例的方便程度	4.5	3.5
对测试人员技能的要求	3	4.5
测试用例的可读性	2	4.5
统一配置管理	1	5
易于调试	3	4
Hudson集成	1	5
价格	1	5

1.3 自动化测试做到什么样算好

自动化测试不是万能的，没有自动化测试是万万不能的。很多公司在看自动化测试的时候都追求自动化率，认为达到 100%的自动化率就是最好的，更有甚者认为自动化率高了，就可以节省人力。恰恰相反，自动化率高了，会使更多的人力投入在维护的成本上，因为系统的需求是在不断变化的，每一个变化都会导致自动化测试案例需要更新调整。

通常来说，人工测试是最基本的，可以做到 100%，而自动化测试更像一件“防弹衣”，用来防护身体的主要部位，有人见过可以 100%包住身体的防弹衣么？那应该就不是防弹衣了，是宇航服。防弹衣在保护我们的同时不影响我们的活动，如果真的包成跟穿着宇航服似的，就会很笨重，进而影响我们的活动。

所以，自动化测试做到什么样算好，也要结合前面的分层来看对于 UI 层面的自动化

测试，保证少量必要的主流程即可，没必要追求自动化率越高越好；Service 层面的接口自动化测试，可以考虑覆盖大部分的流程；Unit 层面的单元测试，这个我倒是觉得做到 100% 是最好的，因为这个的“粒度”已经是最细的层面了，即使有需求变化，一般也很少影响到已有的自动化测试案例，通常都是补充新案例，个别情况可能是废弃旧案例。

其实和写程序类似，写新的程序都很容易，改程序很麻烦。同理，改自动化测试案例也是一件很麻烦的事情，特别是改别人写的自动化测试案例，有时候你会觉得还不如重新写一个。

1.4　小结

自动化测试是一个让人又爱又恨的东西，一味地追求覆盖率，并不一定能带来很大的价值。笔者更认可自动化测试是“防弹衣”的比喻，也希望大家在实际操作中自己来体会这一点，不要让“防弹衣”变成“宇航服”。

2

第 2 章

Robot Framework

本章主要介绍 Robot Framework 的框架特性、如何安装，以及基础知识。

本章的测试案例参考地址：

https://github.com/qitaos/rf-demos/tree/master/rf-book- case/test

2.1 框架介绍

Robot Framework 框架是一个通用的测试框架，一直是由诺西网络（Nokia Siemens Networks Oyj）的人员在维护的。后来诺基亚卖掉了手机业务后，现在更名为诺基亚解决方案网络（Nokia Solutions and Networks）了。

下面介绍一下它的特性：

- 易于使用，采用表格式语法，统一测试用例格式；

- 可以使用关键字驱动（keyword-driven）、数据驱动（data-driven）和行为驱动开发（BDD）完成；
- 重用性好，可以利用现有关键字来组合新关键字；
- 结果报告和日志采用 HTML 格式，易于阅读；
- 平台和应用无关联性；
- 模块结构支持使用不同的接口来测试应用；
- 易于扩展，提供了简单的 API，用户可以自定义基于 Python 或者 Java 的测试库；
- 易于集成，提供了命令行接口和基于 XML 的输出文件；
- 功能全面，支持 Web 测试（Selenium）、Java GUI 测试、启动线程、Telnet、SSH 等；
- RemoteLibrary 接口可以支持分布测试和使用其他编程语言实现测试库；
- 提供 tag 标签来分类和选择案例执行；
- 内置支持变量 variables，支持不同的环境进行测试。

根据我的理解，整理了一个比较形象的 Robot Framework 框架图，如图 2-1-1 所示。

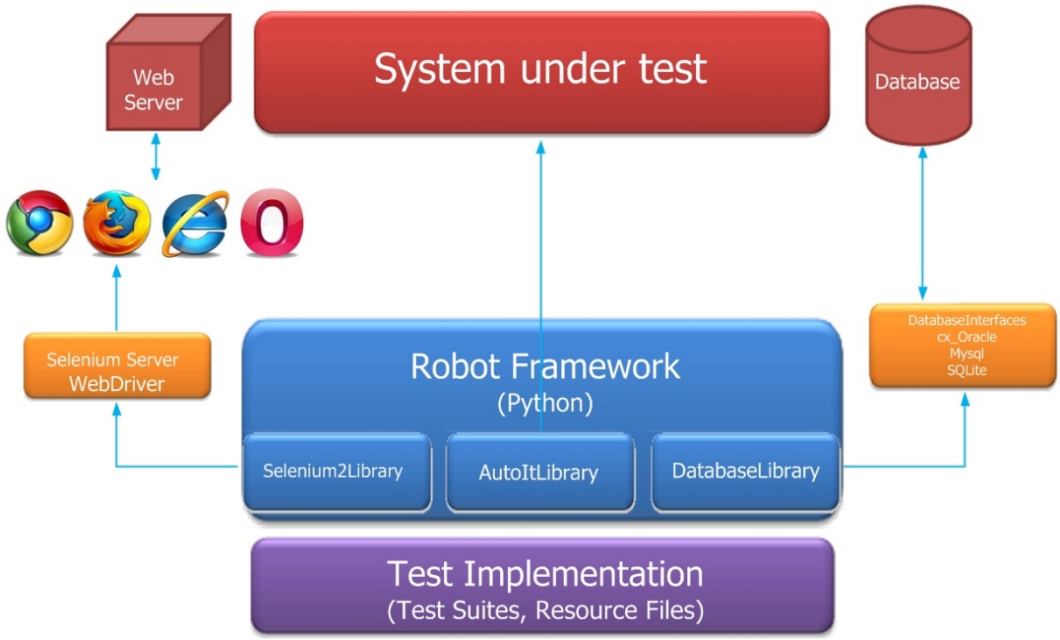

图 2-1-1

Robot Framework 作为框架平台，可以加载各种 Library，通过测试套件、资源文件集

成起来，然后可以针对被测试系统进行不同角度的测试，图 2-1-1 中只列了 3 种 Library，实际上还有很多种不同方向的 Library，这里只是方便大家理解。

2.2 安装指南

2.2.1 Windows 平台

1. Python

官方下载地址：

https://www.python.org/downloads/

首先要安装 Python，这是一切的基础。

打开浏览器，输入官方下载地址，在页面中找到并下载 Python 2.7.9 版本（作者写本书时的最新版本是 2.7.9，由于 Python 版本可能会有更新，大家选择 2.7.x 的版本进行下载即可），选择 Windows x86 MSI installer，文件名 python-2.7.9.msi，如果是 64 位 Windows 操作系统，建议选择 Windows x86-64 MSI installer 下载。

下载后双击文件 python-2.7.9.msi 进行安装，安装路径默认为 C:\Python27，如果不想安装到 C 盘，建议改成 D:\Python27，想使用其他路径也可以。但是需要注意，路径里不能有中文或者空格。选好路径后，“一路”按“Next”按钮，最终完成安装。

Python 安装完成后，必须先要在环境变量 PATH 中加入 Python 的两个路径，例如安装在 D:\Python27，一般建议在 PATH 中增加 D:\Python27 和 D:\Python27\Scripts 两个路径。

Tips：如果 PATH 里有的路径有空格，请把 Python 路径放在有空格路径的前面。建议大家在单独的用户环境变量中新增一个 PATH 变量。

添加环境变量主要是为了后续在 cmd 命令行窗口里安装相关工具能够直接使用 Python 命令安装，如果已经打开过 cmd 命令行窗口，请在设置好环境变量后重新打开 cmd 命令行窗口。

2. Setuptools

官方下载地址：

https://pypi.python.org/pypi/setuptools/

这是一个用于支持安装各种测试库的工具，它会把安装好的测试库的信息放在一个公共的文件里（easy-install.pth），算是一个测试库的引导清单，作用非常大（此处先不细说）。此外它的 easy_install 工具是用来安装 egg 文件的（后面遇到此类文件时再说）。

打开浏览器，输入官方下载地址，下载一个最新的版本（作者写本书时的最新版本是 setuptools-14.0）。单击页面上的“Download”按钮，跳转到页面下方会有 3 个文件，选择 tar.gz 或 zip 文件都可以（我一般下载 tar.gz 文件）。

安装方法：解压缩 tar.gz 文件，在 cmd 命令行窗口进入解压出来的目录，输入 python setup.py install，然后按回车键等待安装完成。

可能有些读者第一次学习，这里详细地描述一下安装过程，如果是已经熟悉的读者，可以跳过。

以下载 setuptools-14.0.tar.gz 文件为例，文件下载完成后，将它解压缩，用鼠标右键单击文件，选择“解压到当前文件夹”命令。如果选择“解压到 setuptools-14.0”命令，会导致多嵌套了一层目录，后续安装时要多写一层目录。

此时打开 cmd 命令行窗口，进入解压缩出来的目录，例如 setuptools-14.0.tar.gz 在 D:\soft 目录下，那么就在命令行窗口输入 cd D:\soft\setuptools-14.0\，然后按回车键（如果你的文件在其他目录，请修改为对应的目录），这个目录里有个名叫 setup.py 的文件，可以输入 dir 命令查看一下是否存在（如果你之前选择解压到 setuptools-14.0，这里就没有 setup.py 文件了，你要再进入下一层子目录才有这个文件），确认当前路径是正确的，接着在 cmd 命令行窗口输入 python setup.py install，然后按回车键，等待安装完成。

后面此类文件的安装基本相同，因此后续只列安装方法，不再进行详细描述。

3. Robot Framework

官方下载地址：

https://github.com/robotframework/robotframework/releases

怀旧下载地址：

http://code.google.com/p/robotframework/downloads/list

这就是本书的核心——自动化框架 Robot Framework，官方网站地址：http://robotframework.org/，此外介绍两个不错的网站，一个是我们 QQ 群（247870083）的管理员陈高敏（葡萄）创建的：http://robotframework.net/；另一个是徐毅老师创建的：http://robotframework.cn/。

下载路径提供了两个，原先 Robot Framework 一直放在 google code 上，后来 Google Code 计划关闭，众多项目纷纷转移，RF 也转移到了 Github 上。在本书写作期间，Google 再次公布了 Google Code 的关闭计划，作为留念，保留了一下 Google 的下载地址，或许未来读到本书的朋友已经无法访问到“怀旧”地址了，读者朋友下载时请到 Github 的地址下载。

打开浏览器，输入官方下载地址，找一个最新的版本下载（作者写本书时的最新版本是 Robot Framework 2.8.7），单击下载链接会自动下载安装包 robotframework-2.8.7.tar.gz。

安装方法：解压缩 tar.gz 文件，在 cmd 命令行窗口进入解压出来的目录，输入 python setup.py install，然后按回车键等待安装完成。

4. wxPython

官方下载地址：

http://www.wxpython.org/download.php

备用下载地址：

http://sourceforge.net/projects/wxpython/files/wxPython/2.8.12.1/

wxPython 是用于支持 Python 图形化界面的，安装它主要是用来运行 RIDE 的。

官方下载页面放的都是最新的 3.0 版本，但是 RIDE 本身没说过它是否支持 3.0，我特意尝试了一下安装 3.0 的版本，安装后运行 RIDE 会报如下错误信息：

```
   D:\>ride.py
   Wrong wxPython version.
   You need to install wxPython 2.8 toolkit with unicode support to run RIDE.
   wxPython 2.8.12.1 can be downloaded from http://sourceforge.net/projects/
wxpython/files/wxPython/2.8.12.1/
```

错误信息里提示必须安装 wxPython 2.8 的 unicode 版本才能使用，特意尝试了一下 2.9 版本，也是一样不支持。所以我一直在用这个比较旧的 wxPython 版本，文件名是 wxPython2.8-win32-unicode-2.8.12.1-py27.exe，那就要到它的托管网站 sourceforge 上下载了，如图 2-2-1 所示。也就是备用下载地址，进入页面后会有很多版本供你选择，选择 2.8.12.1 版本进入，里面有很多文件，由于文件名比较长，大家在选择时可以把鼠标放在下载链接上不动，看一下提示出来的文件名，注意文件名里的关键信息“win32 + unicode + py27”，如果是使用 64 位操作系统的同学，下载时就找“win64 + unicode + py27”。

下载完成后，运行程序，“一路”单击“Next”按钮，最后完成安装即可。

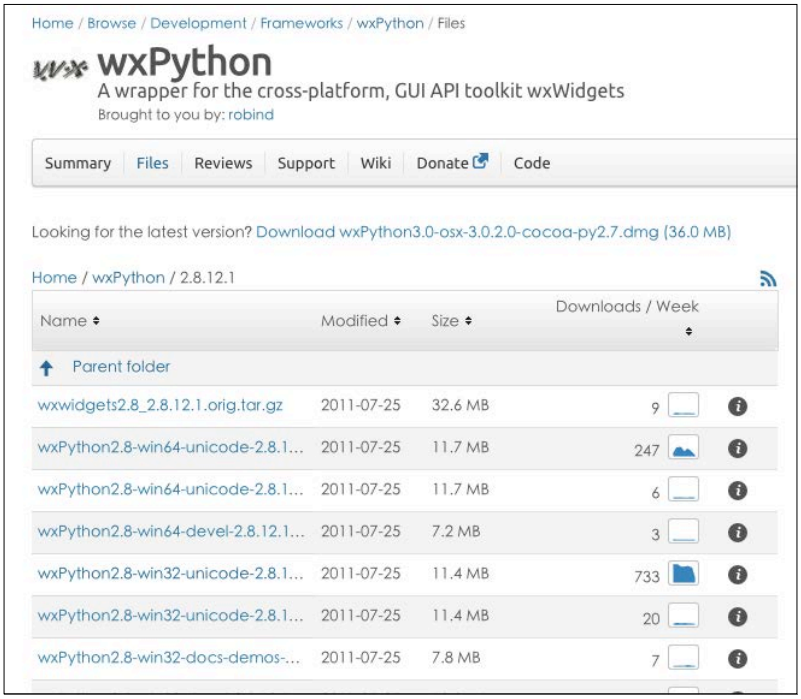

图 2-2-1

5. robotframework–ride

官方下载地址：

https://github.com/robotframework/RIDE/releases

怀旧下载地址：

http://code.google.com/p/robotframework-ride/downloads/list

RIDE 是一个可视化界面工具，专用于大家编写测试案例，也有其他工具可以来写案例，但是我一直习惯用 RIDE，特别是对于初学者来说，RIDE 方便很多。

前面提到众多项目从 Google Code 上迁移走，RIDE 也是其中之一，而对 Windows 的用户来说，不是好事，因为以前在 Google Code 上，下载列表里会有 exe 安装程序，这个程序会在桌面增加一个机器人图标的快捷方式，直接双击图标就可以打开 RIDE 了，如图 2-2-2 所示。

图 2-2-2

打开怀旧下载地址，可以看到这样的界面，如图 2-2-3 所示。

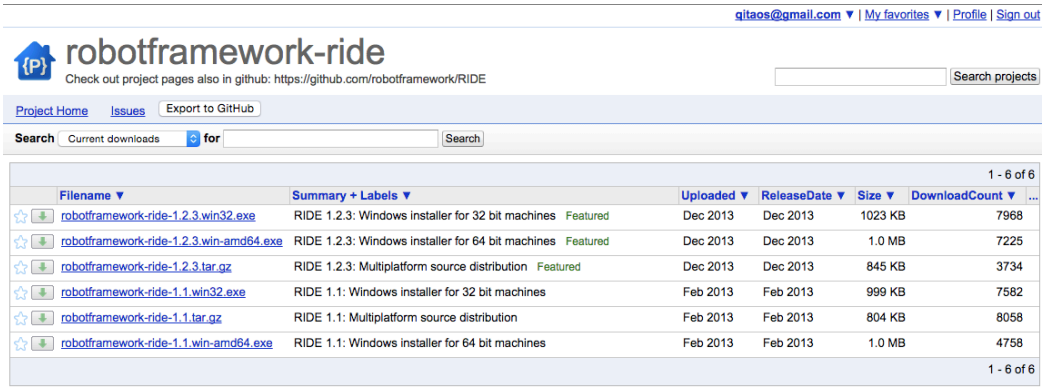

图 2-2-3

这里最新的版本是 robotframework-ride-1.2.3.win32.exe，但是实际上 RIDE 最新版已经到 1.3，在 Github 上存放，但是没有 exe 安装包，只有 tar.gz 和 zip 文件，大家可以访问官方下载路径来下载最新版本。

安装方法：解压缩 tar.gz 文件，在 cmd 命令行窗口进入解压出来的目录，输入 python setup.py install，按回车键等待安装完成。

当然，不一定非要快捷方式图标才能打开 RIDE，也可以进入 cmd 命令行窗口输入 ride.py 来打开 RIDE。

如果想要图标，有两个办法：第一个办法是先下载 1.2.3 的 exe 文件，安装成功后再安装 1.3 版本；第二个办法是直接安装 1.3 版本，然后手动添加快捷方式图标。

手动添加快捷方式图标的方法是，先在桌面上单击鼠标右键，在弹出的快捷菜单中选择“新建→快捷方式”命令，然后在“请键入对象的位置”的空白处，输入命令 D:\Python27\pythonw.exe -c “from robotide import main; main()”，如图 2-2-4 所示。

单击“下一步”按钮，在“键入该快捷方式的名称”的文本框中，输入名称 RIDE，如图 2-2-5 所示。

单击“完成”按钮，完成手动添加快捷方式图标，如图 2-2-6 所示。

但是创建完会发现，图标不是机器人。这时在新建的快捷方式的图标上，单击鼠标右键，选择“属性”命令，会弹出“RIDE 属性”对话框，如图 2-2-7 所示。

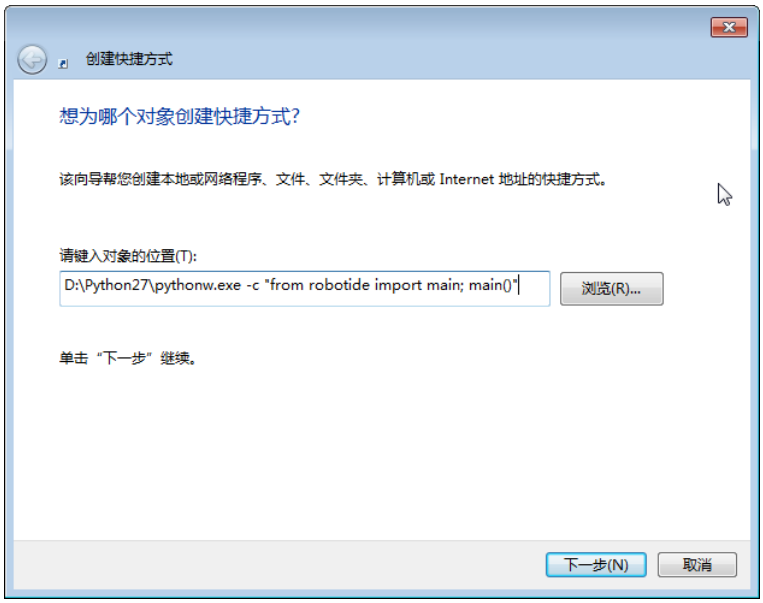
创建快捷方式
想为哪个对象创建快捷方式?
该向导帮您创建本地或网络程序、文件、文件夹、计算机或 Internet 地址的快捷方式。
请键入对象的位置(T):
D:\Python27\pythonw.exe -c "from robotide import main; main()"
浏览(R)...
单击“下一步”继续。
下一步(N)
取消

图 2-2-4

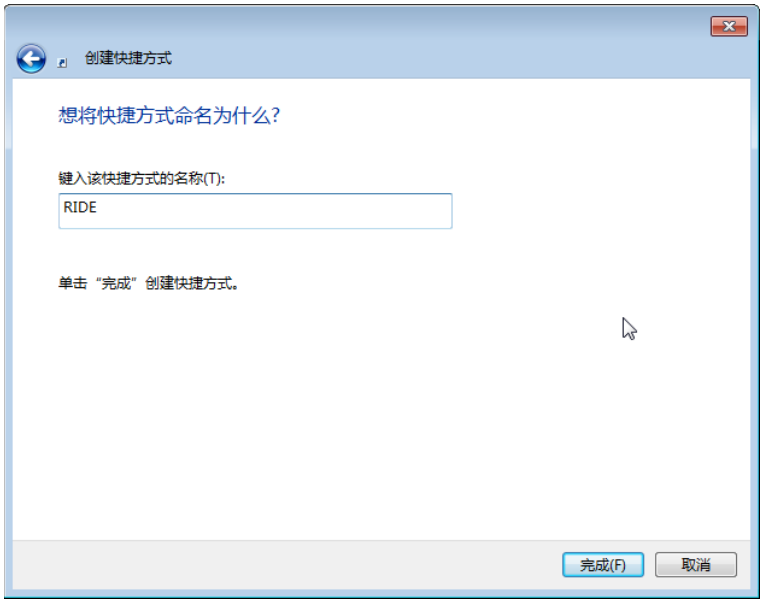
创建快捷方式
想将快捷方式命名为什么?
键入该快捷方式的名称(T):
RIDE
单击“完成”创建快捷方式。
完成(F)
取消

图 2-2-5

图 2-2-6

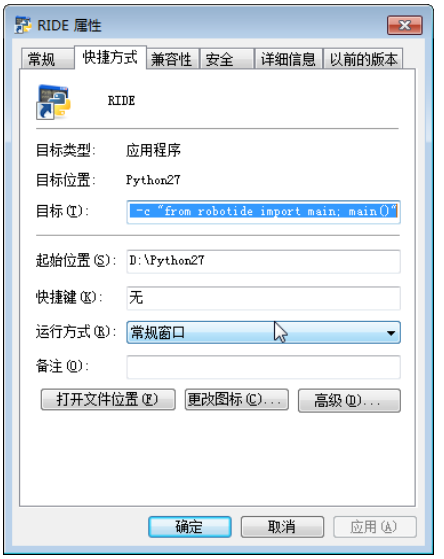

图 2-2-7

单击“更改图标”按钮，弹出“更改图标”对话框，找到目录 D:\Python27\Lib\site-packages\robotide\widgets，里面有个名叫 robot.ico 的图标，如图 2-2-8 所示。

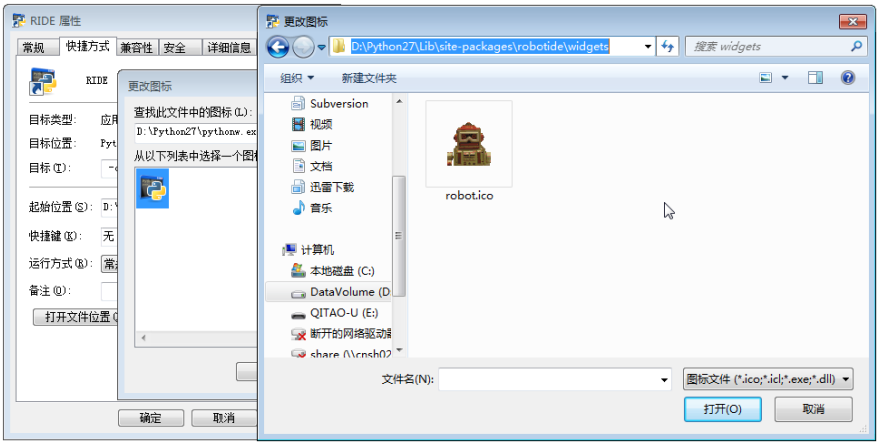

图 2-2-8

选中此图标，单击“打开”按钮，然后单击“确定”按钮，再单击“确定”按钮，效果如图 2-2-9 所示。

图 2-2-9

6. 小结

如果是 64 位的 Windows 操作系统，那么从 Python 到后面所有的工具都要安装 64 位的。其实安装 32 位的工具也可以，但是个别测试库可能会有问题，所以还是安装 64 位的工具比较好。中间有一些工具不区分 32 位或 64 位，那么这些工具是 32 位和 64 位都兼容的。

总结一下 Windows 的安装说明：

（1）如果是 exe 或 msi 的，直接双击安装即可；

（2）对于只有源代码的测试库（tar.gz 或 zip 文件）的安装，请在进入 cmd 命令行窗口后，进入测试库的目录（有 setup.py 文件的目录），输入 Python setup.py install，然后进行安装；如果提示 python 不是命令，请把 Python 的安装目录加到环境变量的 PATH 中；

（3）对于只有 egg 文件的测试库的安装，需要先安装 setuptools，然后在 cmd 命令行窗口使用 easy_install 加 egg 文件名或目录名，例如 easy_install docutils-0.9-py2.7.egg。

第一种情况是直接安装，第二种和第三种情况是命令行安装，这两个命令行安装的方式，必须先要在环境变量 PATH 中加入 Python 的两个路径。如果你的 Python 安装在 D:\Python27，那么就增加 D:\Python27 和 D:\Python27\Scripts 两个路径到环境变量 PATH 中。

以上所讲的 5 点是最基础的 RF 安装，保证可以运转起来，后续在各个不同测试库的地方，再介绍单独的安装方法，但是都要在这前 5 点做完之后。

2.2.2　Mac 平台（Linux 平台可参考）

Mac 上需要安装的工具和 Windows 一样，工具介绍在 Windows 里已经介绍过了，这里主要介绍安装方法。

1. Python

官方下载地址：

https://www.python.org/downloads/

其实 Mac 上本身已经安装好了 Python，你可以在自己 Mac 的终端里输入 python --version（注意是 2 个“-”）来查看 Python 的版本。

所以这一步实际上可以跳过，如果你想安装最新版本的 Python，那就到官方下载地址去下载 Mac 版本的安装包，最新版在首页上直接就有下载链接，安装包是 pkg，下载后直接双击文件安装即可。如果你想找指定版本的安装包，那就要到“downloads”下的 Mac OS X 页面里找你需要的版本。

2. Setuptools

官方下载地址：

https://pypi.python.org/pypi/setuptools/

打开浏览器，输入官方下载地址的 URL，找一个最新的版本下载，和 Windows 那里一样，下载 tar.gz 文件。

安装方法：解压缩 tar.gz 文件，在 cmd 命令行里进入解压出来的目录，输入 sudo python setup.py install，然后按回车键等待安装完成。

需要注意，如果直接像 Windows 一样用 python setup.py install 安装，可能会提示权限不足，此时要在命令前面加上 sudo 提权，输入 sudo python setup.py install 然后按回车键，输入管理员密码才能安装成功。所以 Mac 中此类安装都请加上 sudo。

3. Robot Framework

官方下载地址：

https://github.com/robotframework/robotframework/releases

怀旧下载地址：

http://code.google.com/p/robotframework/downloads/list

打开浏览器，输入官方下载地址的 URL，找一个最新的版本下载（作者写本书的时候最新版本是 Robot Framework 2.8.7），单击链接会自动下载安装包 robotframework-2.8.7.tar.gz。

安装方法：解压缩 tar.gz 文件，在 cmd 命令行里进入解压出来的目录，输入 sudo python setup.py install，然后按回车键等待安装完成。

4. wxPython

官方下载地址：

http://www.wxpython.org/download.php

备用下载地址：

http://sourceforge.net/projects/wxpython/files/wxPython/2.8.12.1/

wxPython 是用于支持 Python 图形化界面的，安装它主要是用来运行 RIDE 的。

由于官方下载页面只有最新的 3.0 版本，但是前面 Windows 平台安装 wxPython 时，已经说了它不支持 3.0 版本，但是笔者还是尝试安装了 3.0 版本，在安装或运行后面的 RIDE 时会提示如下信息：

Wrong wxPython version.

You need to install wxPython 2.8 or 2.9 toolkit with unicode support to run RIDE.

wxPython 2.8.12.1 can be downloaded from http://sourceforge.net/projects/wxpython/files/wxPython/2.8.12.1/

和 Windows 不同的是，这里写了“or 2.9”的 unicode 的版本，为了试验一下 2.9 版本是不是可以用，笔者去找了一下 2.9 版本（访问备用地址的页面，页面上有个“Parent folder”的链接，单击此链接后，可以看到很多版本），在 2.9 版本开始，支持 Mac OS X 的 py27 安装文件有两个文件，至少笔者从文件名上看不到有 unicode 字样了，一个带有 carbon 标识，一个带有 cocoa 标识。前者是为了兼容 Mac OS X 10.5 以及以前的版本，如果你的 Mac 是 OS X 10.5 以后的版本，那么选择 cocoa 版本的链接下载。笔者特意尝试了 2.9 的几个版本，运行后面的 RIDE，不会说是 wxPython 版本问题了。但是在笔者这里，无论是 carbon 还是 cocoa 的 2.9 版本，安装后运行 RIDE，可以打开 RIDE 界面，然后就异常退出了，笔者猜想还是因为没有 unicode 的原因吧，难怪报错信息里没有写 2.9 的具体哪个版本的下载地址。

所以还是老老实实地用 2.8 版本吧，就用提示信息里说的 2.8.12.1 版本，文件名 wxPython2.8-osx-unicode-2.8.12.1-universal-py2.7.dmg，步骤和 Windows 一样，打开浏览器进入备用下载地址，找到和笔者提供的名字一样的 dmg 文件进行下载。

下载完成后，双击 dmg 文件，会打开一个窗口，里面有 pkg 文件，双击 pkg 文件进行安装就可以了。

5. robotframework–ride

官方下载地址：

https://github.com/robotframework/RIDE/releases

怀旧下载地址：

http://code.google.com/p/robotframework-ride/downloads/list

Mac 上直接去官方下载地址，下载 1.3 的 tar.gz 文件。

安装方法：解压缩 tar.gz 文件，在 cmd 命令行里进入解压出来的目录，输入 python setup.py install，按回车键等待安装完成。

通常在第一次安装后，在终端输入 ride.py 回车，会提示 Python 要运行在 32 位模式下：“python should be executed in 32-bit mode to support wxPython on mac. Check BUILD.rest for details”。

因为 Mac 默认是 64 位运行，所以 Python 也默认运行 64 位，此时需要调整一下默认 Python 为 32 位，有两种操作方式。

（1）在终端里执行下面这句：

defaults write com.apple.versioner.python Prefer-32-Bit -bool yes

（2）或者在“~/.bash_profile”里增加下面这句：

export VERSIONER_PYTHON_PREFER_32_BIT=yes

保存退出后运行 source ~/.bash_profile 后就可以正常打开 RIDE 了。

6. 小结

总结一下安装说明：

（1）如果是 dmg 或 pkg，直接双击安装即可；

（2）对于只有源代码的测试库（tar.gz 包）的安装，请在进入终端命令行后，进入测试库的目录（有 setup.py 的目录），输入 sudo python setup.py install 进行安装；

（3）对于只有 egg 文件的测试库的安装，需要先安装 setuptools，然后在终端命令行使用 easy_install 加 egg 文件名或目录名，例如 easy_install docutils-0.9-py2.7.egg，如果提示没权限，记得加 sudo。虽然前面没有碰到，但是不代表以后没有。

第一种情况是直接安装，第二种和第三种情况是命令行安装，对于 Mac 来说，不需要额外添加环境变量就可以直接运行了。

上面的 5 点是最基础的 RF 安装，保证可以运转起来，后续在各个不同测试库的时候再介绍单独的安装方法，但是都要在这前 5 点做完后。

2.2.3　pip 安装

官方下载地址：

https://pypi.python.org/pypi/pip

前面介绍的 Windows 和 Mac 上的命令行安装，大多数都是基于 setuptools 的，setuptools 这种安装方式需要先下载好安装包，然后在本地安装。其实这也是笔者个人比较推荐的，因为在笔者接触 Robot Framework 最初，公司网络有很多限制，无法进行在线安装，所以有网络困扰的同学，请选择前面的安装方式。如果网络比较好的也可以用 setuptools，因为如果你要安装的库有一些依赖库，setuptools 会帮你下载安装的。

此外还有一种 pip 安装方式，网上很多同学也比较推荐，需要先安装 pip。

打开浏览器访问官方下载地址，找到 pip 最新的 tar.gz 文件并进行下载，安装方法和前面介绍过的一样，先解压缩，然后用 python setup.py install 安装。

然后可以用 pip install LibraryName 进行相应测试库的在线安装，不需要本地提前下载安装包，pip 会自动从 pypi.python.org 上搜索安装包并下载安装。

举个例子，如果想安装 Robot Framework，可以在命令行输入 pip install robotframework 进行安装。完整的 pip 命令请在命令行输入 pip --help（注意是 2 个“-”）查看。

这里不详细介绍了，还是推荐大家使用前面两个小节介绍的方法，对 pip 感兴趣，想深入了解 pip 的同学，可以自行学习。

2.3　第一个案例

为了让读者有个直观的认识，先抛开概念性的内容，从实操入手。建议大家跟着一起做。

2.3.1 打开 RIDE

在 Windows 平台下，直接双击桌面的“Robot 机器人”图标即可，或者进入到 Python 目录的 scripts 子目录下，双击 ride.py 文件。也可以在命令行输入 ride.py，然后按回车键，就可以看到 RIDE 界面，如图 2-3-1 所示。

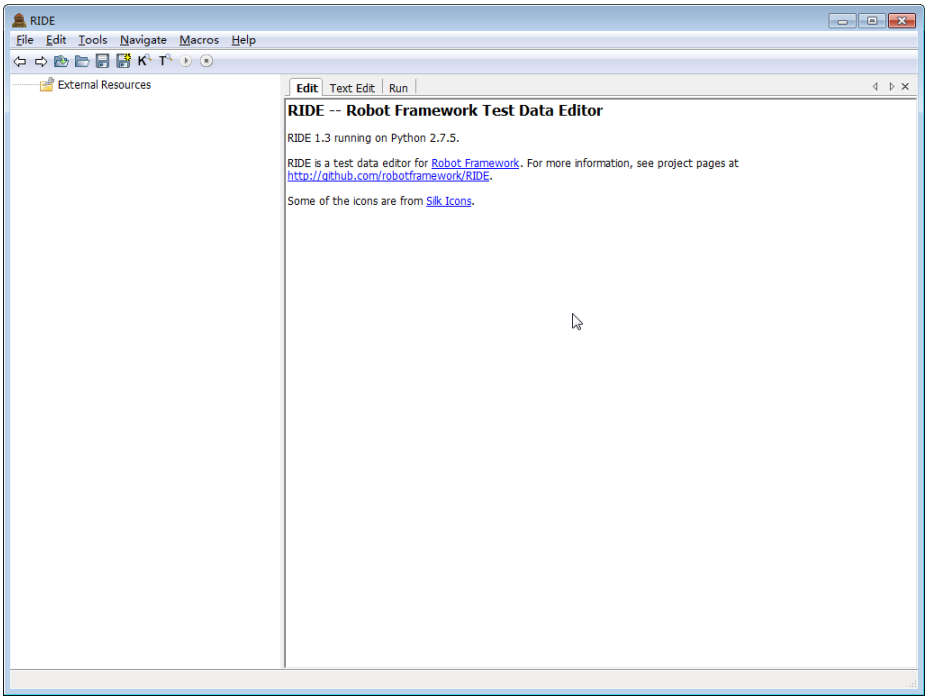

图 2-3-1

在 Mac 平台下，直接在终端命令行输入 ride.py，然后按回车键，就可以看到如图 2-3-2 所示界面。

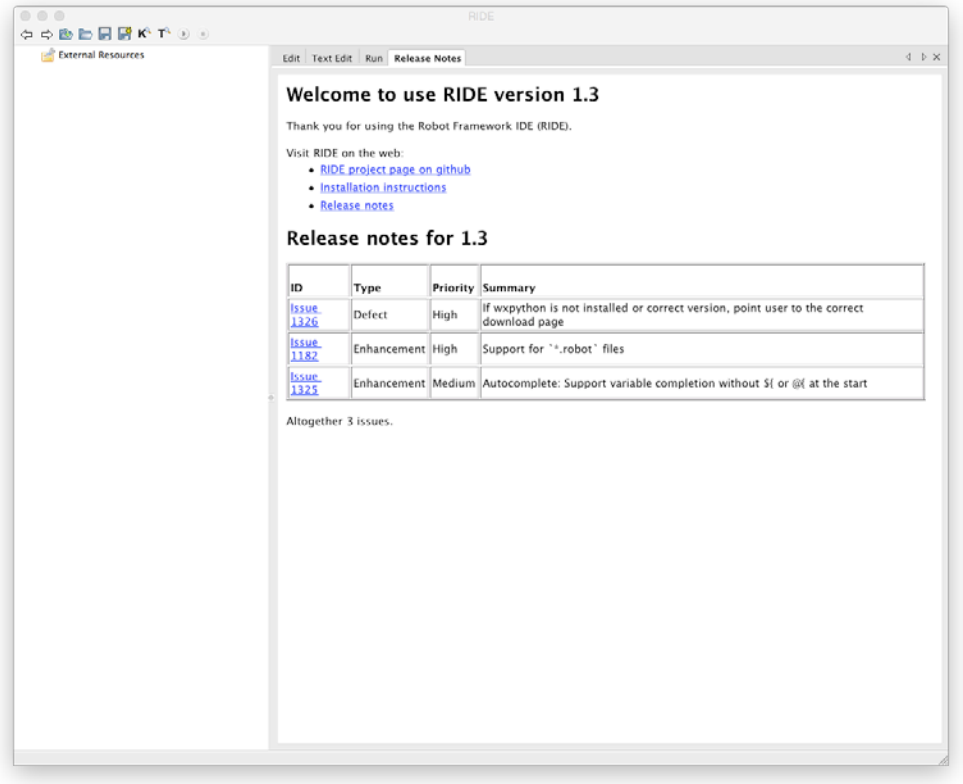

图 2-3-2

由于在 Mac 和 Windows 操作系统上基本一致，所以本节主要以 Windows 的截图为主。

2.3.2　创建工程和测试套

选择“File→New Project”命令，如图 2-3-3 所示。

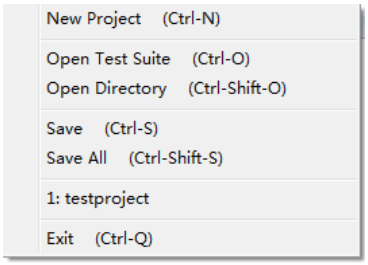

图 2-3-3

弹出“New Project”对话框，在“Name”文本框中输入一个名称，例如 testproject，在右侧选中“Directory”单选按钮，如图 2-3-4 所示。

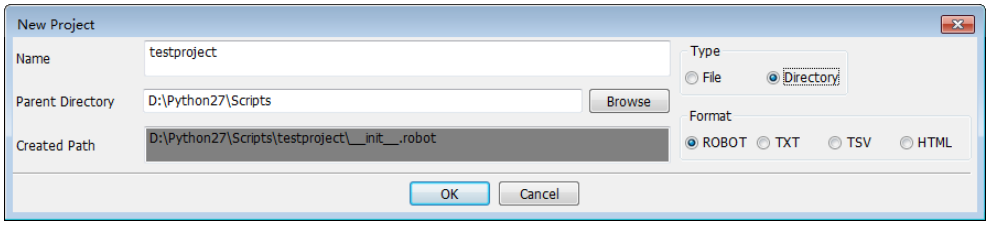

图 2-3-4

然后单击“OK”按钮，弹出如图 2-3-5 所示界面。

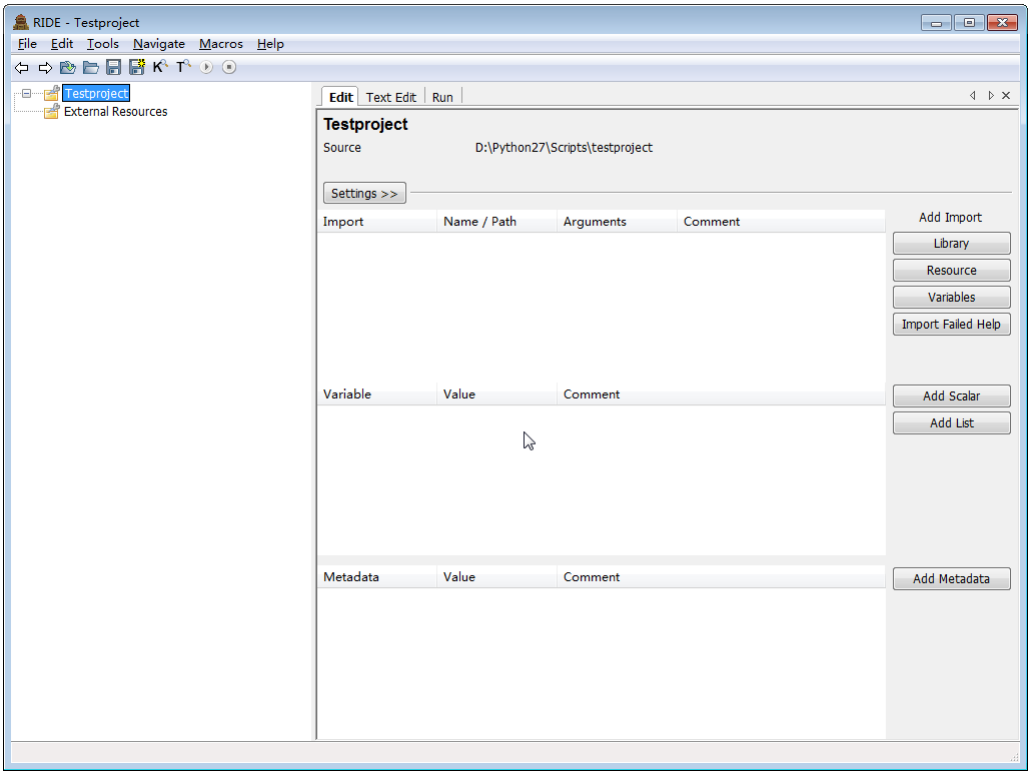

图 2-3-5

用右键单击“Testproject”选项，在弹出的菜单中单击“New Suite”命令，弹出“Add Suite”对话框，在“Name”文本框中输入一个名称，例如 testsuite1，如图 2-3-6 所示。

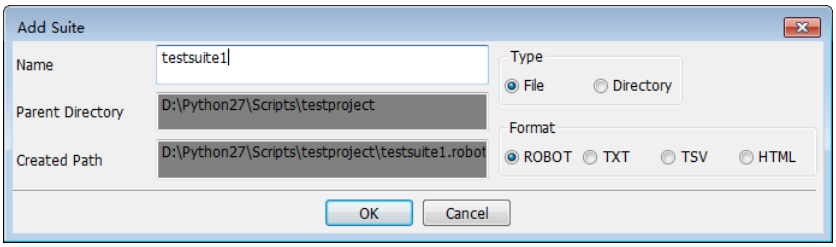

图 2-3-6

然后单击“OK”按钮，弹出如图 2-3-7 所示界面。

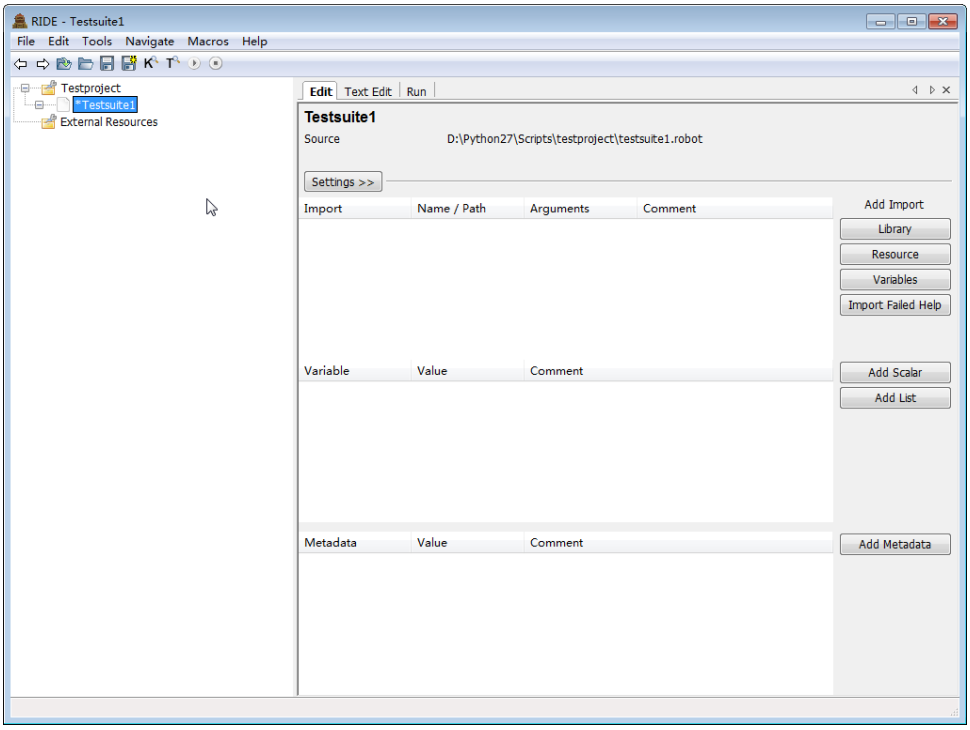

图 2-3-7

2.3.3　创建案例

用鼠标右键单击“testsuite1”选项，在弹出的菜单中单击“New Test Case”命令，弹出“New Test Case”对话框，在“Name”文本框中输入一个名称，例如 case1，如图 2-3-8 所示。

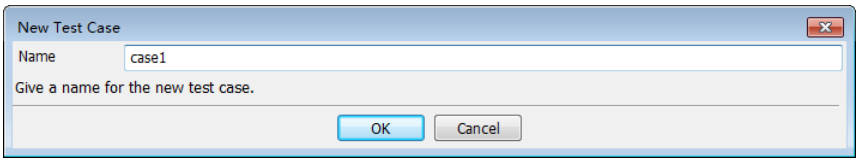

图 2-3-8

然后单击“OK”按钮，弹出如图 2-3-9 所示界面。

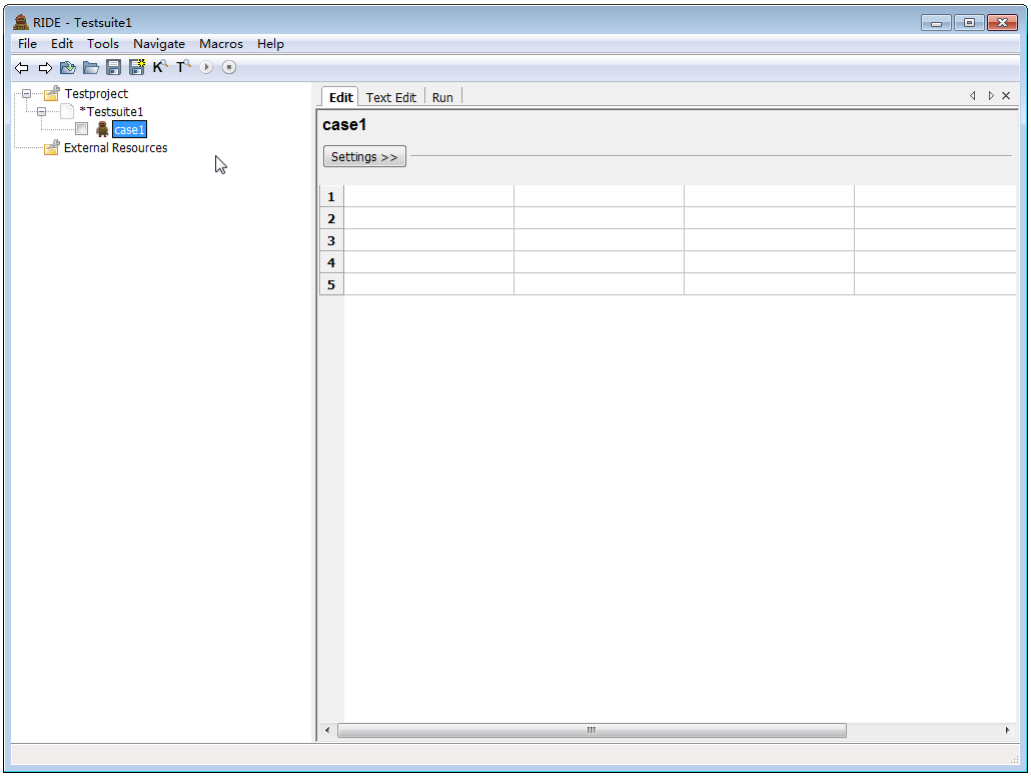

图 2-3-9

2.3.4 写一行脚本

第一个脚本就用 log 命令，这个命令是用来打印日志的，对比 C 语言，你可以理解为“printf”，我们来打印一个“hello world”。

脚本写完后，如图 2-3-10 所示。

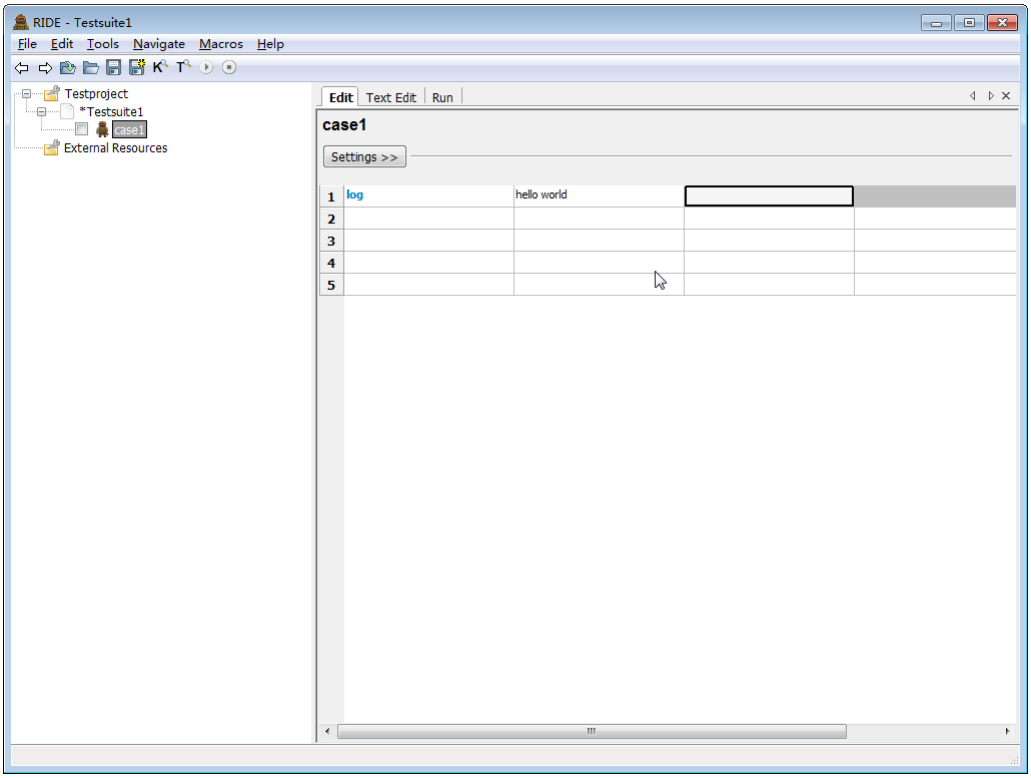

图 2-3-10

2.3.5　运行案例

单击工具栏像播放按钮的图标运行案例，运行结果如图 2-3-11 所示。

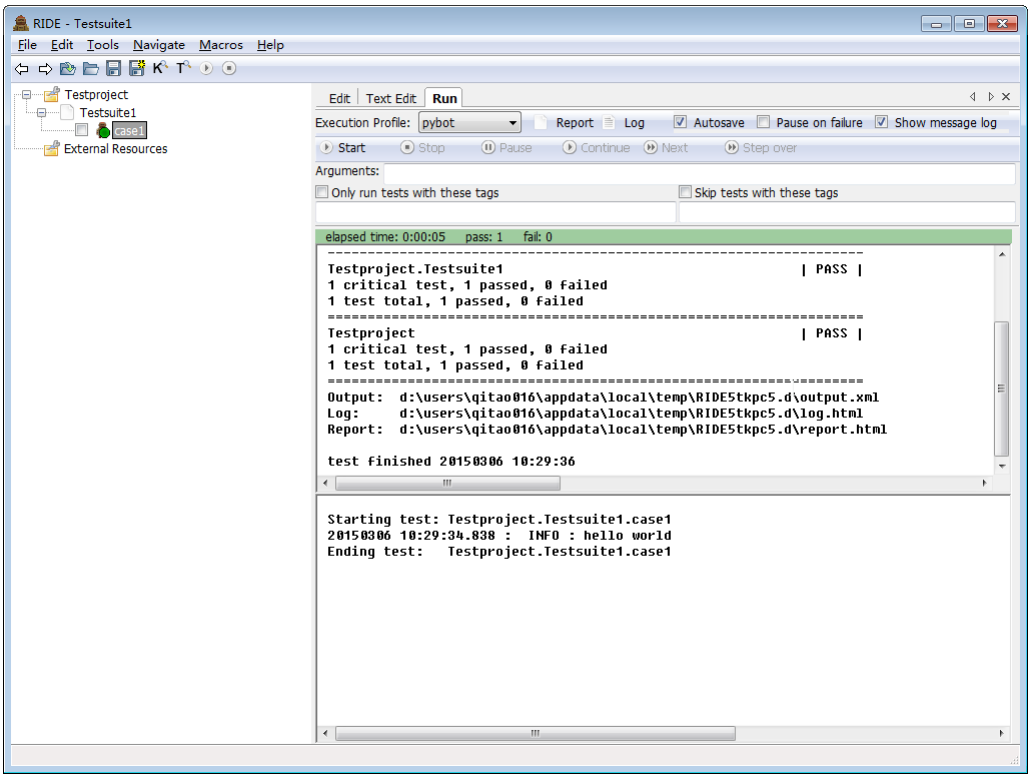

图 2-3-11

2.3.6 查看运行结果

单击 Report 或 Log 图标查看报告，Report 和 Log 分别是两种报告，展示的信息不一样，Report 报告如图 2-3-12 所示。Log 报告如图 2-3-13 所示。

Report 报告主要是概括性的报告，总体案例执行情况。Log 报告则是更为详细的案例步骤的报告。

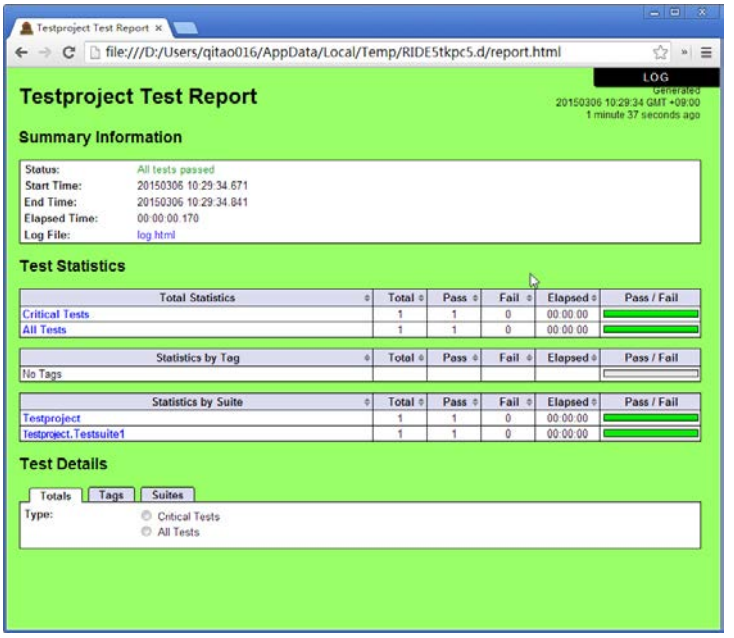
Testproject Test Report
file:///D:/Users/qitao016/AppData/Local/Temp/RIDE5tkpc5.d/report.html
LOG
Testproject Test Report
Generated
20150306 10:29:34 GMT +09:00
1 minute 37 seconds ago
Summary Information
Status: All tests passed
Start Time: 20150306 10:29:34.671
End Time: 20150306 10:29:34.841
Elapsed Time: 00:00:00.170
Log File: log.html
Test Statistics
Total Statistics | Total | Pass | Fail | Elapsed | Pass / Fail
Critical Tests | 1 | 1 | 0 | 00:00:00
All Tests | 1 | 1 | 0 | 00:00:00
Statistics by Tag | Total | Pass | Fail | Elapsed | Pass / Fail
No Tags
Statistics by Suite | Total | Pass | Fail | Elapsed | Pass / Fail
Testproject | 1 | 1 | 0 | 00:00:00
Testproject.Testsuite1 | 1 | 1 | 0 | 00:00:00
Test Details
Totals
Tags
Suites
Type:
Critical Tests
All Tests

图 2-3-12

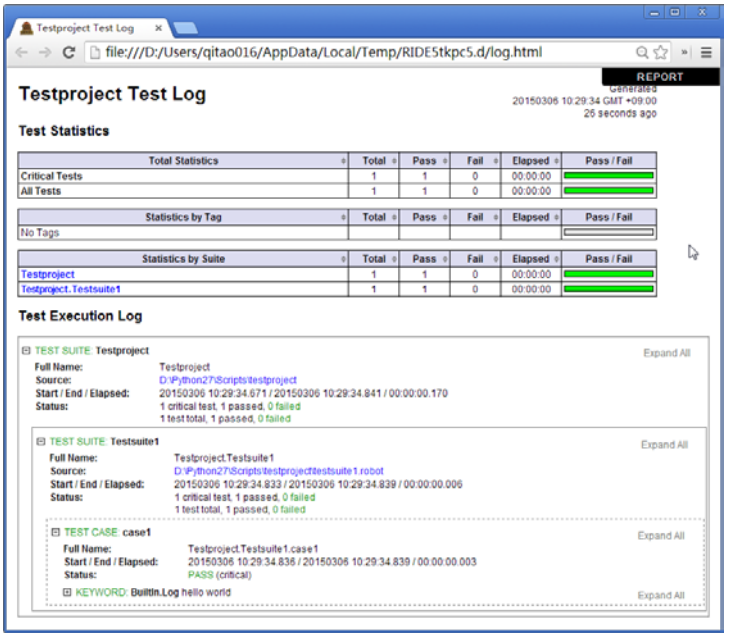
Testproject Test Log
file:///D:/Users/qitao016/AppData/Local/Temp/RIDE5tkpc5.d/log.html
REPORT
Testproject Test Log
Generated
20150306 10:29:34 GMT +09:00
26 seconds ago
Test Statistics
Total Statistics | Total | Pass | Fail | Elapsed | Pass / Fail
Critical Tests | 1 | 1 | 0 | 00:00:00
All Tests | 1 | 1 | 0 | 00:00:00
Statistics by Tag | Total | Pass | Fail | Elapsed | Pass / Fail
No Tags
Statistics by Suite | Total | Pass | Fail | Elapsed | Pass / Fail
Testproject | 1 | 1 | 0 | 00:00:00
Testproject.Testsuite1 | 1 | 1 | 0 | 00:00:00
Test Execution Log
TEST SUITE: Testproject
Expand All
Full Name: Testproject
Source: D:\Python27\Scripts\testproject
Start / End / Elapsed: 20150306 10:29:34.671 / 20150306 10:29:34.841 / 00:00:00.170
Status: 1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
TEST SUITE: Testsuite1
Expand All
Full Name: Testproject.Testsuite1
Source: D:\Python27\Scripts\testproject\testsuite1.robot
Start / End / Elapsed: 20150306 10:29:34.833 / 20150306 10:29:34.839 / 00:00:00.006
Status: 1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
TEST CASE: case1
Expand All
Full Name: Testproject.Testsuite1.case1
Start / End / Elapsed: 20150306 10:29:34.836 / 20150306 10:29:34.839 / 00:00:00.003
Status: PASS (critical)
KEYWORD: BuiltIn.Log hello world
Expand All

图 2-3-13

2.3.7 小结

这样我们已经完成了第一个案例，也运行了案例，查看了结果。可能很多细节你不太明白，不用着急，后面的章节会一一道来。

2.4 RIDE 工具

2.4.1 工具介绍

本节开始主要介绍 RIDE 工具的使用，基本上理解了这些内容，就可以算入门了。

笔者把 RIDE 的界面大致分了 4 个区域：菜单栏、工具栏、工程文件区、工作区，如图 2-4-1 所示。

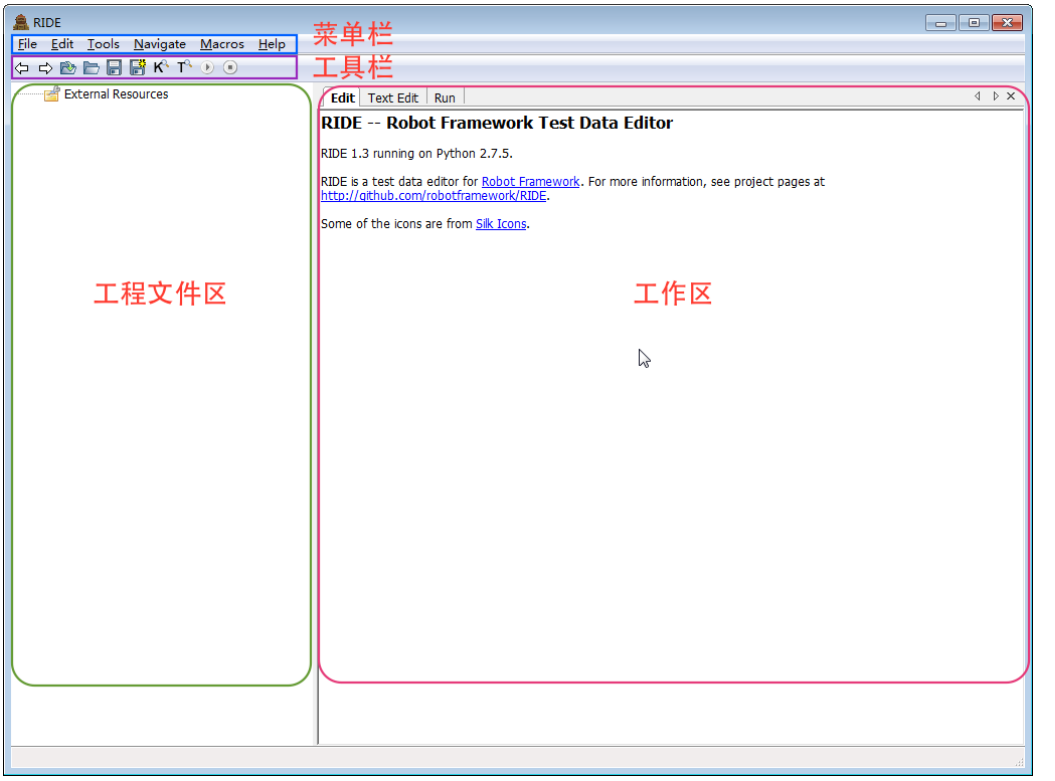

图 2-4-1

- 菜单栏：RIDE 所有的功能都在这里面；
- 工具栏：比较常用的功能，可以快捷操作；
- 工程文件区：这里将会是一个目录一个目录的树形结构（当前是刚打开的样子，里面只有一个空的 External Resources）；
- 工作区：这里是主要编辑案例，运行案例的操作区。

2.4.2　菜单栏&工具栏

其实笔者一直在想，这个工具怎么讲大家更容易理解，最方便的应该是将实际的例子和操作结合起来进行讲解，这个肯定会有。不过笔者觉得还是先要全面介绍一下工具。当然，对于菜单栏和工具栏，只是罗列功能描述，因为后面都会用到的。对于工程文件区和工作区，会在后面以实际案例进行讲解。

在这里介绍菜单栏，同时会把对应的工具栏按钮穿插在其中，因为工具栏实际上就是菜单栏里的菜单项。另外，实际上在工程文件区和工作区其实还有快捷菜单（在区域中用鼠标右键单击），因为太多太杂，就先不整体介绍了，后续章节中会在讲解具体功能时介绍一下。

1. File 文件

Windows 平台的 File 菜单，如图 2-4-2 所示。Mac 平台的 File 菜单，如图 2-4-3 所示。

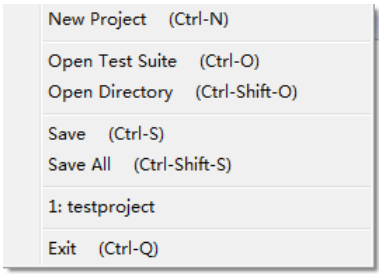

图 2-4-2

图 2-4-3

File 菜单下的主要菜单项，见表 2-4-1。

表 2-4-1

New Project	新建项目
Open Test Suite	打开测试套件文件

续表

New Project	新建项目
Open Directory	打开目录
Save	保存
Save All	保存全部
Exit	退出

在 Save All 和 Exit 中间是你最近打开过的工程列表，这样可以方便地切换工程，最多会显示 4 个工程。

两个 Open 菜单项分别对应工具栏这 2 个图标：  。左边是 Open Test Suite，右边是 Open Directory。和直接单击菜单栏里的菜单项是一样的。

Save 和 Save All，分别对应工具栏这 2 个图标：  。左边是 Save，右边是 Save All。Save 用来保存当前焦点所在的文件，如果你修改了多个文件，用 Save All 可以全部保存。

2. Edit 编辑

Windows 平台的 Edit 菜单，如图 2-4-4 所示。Mac 平台的 Edit 菜单，如图 2-4-5 所示。

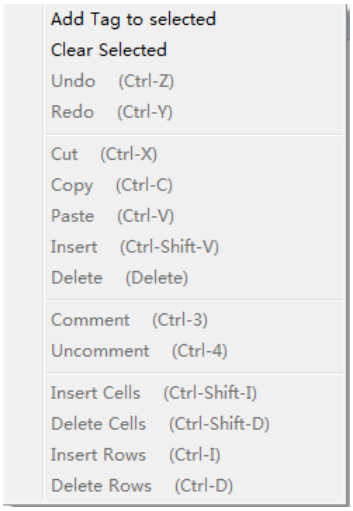

图 2-4-4

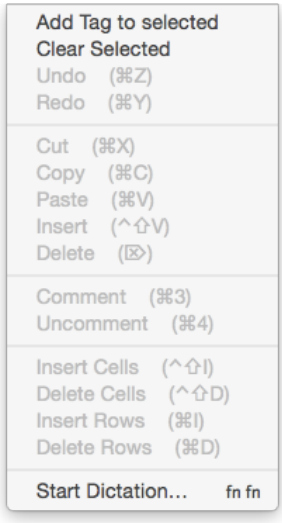

图 2-4-5

Edit 菜单下的主要菜单项，见表 2-4-2。

表 2-4-2

Add Tag to selected	对选中的案例增加Tag
Clear Selected	清除选中的案例
Undo	撤销
Redo	重做
Cut	剪切
Copy	复制
Paste	粘贴
Insert	插入
Delete	删除
Comment	注释当前行
Uncomment	取消注释
Insert Cells	插入单元格
Delete Cells	删除单元格
Insert Rows	插入行
Delete Rows	删除行
Start Dictation	Mac里的启用听写功能

这里很多功能大家应该都很熟悉了，就不详细说了，只说几个特别的。

- Add Tag to selected：要先选中案例，可以选一个或多个，然后用这个命令批量给选中的案例一起添加 Tag 标签。Tag 标签在后续的章节会介绍，它是很有作用的东西；
- Clear Selected：虽然和上一个添加 Tag 挨着，但是这个命令的作用只是把选中的案例变成不选中，也就是清除选中状态，不要以为是清除 Tag；
- Comment：把选中的案例行注释，Uncomment 是取消注释，可以选中多行一起操作；
- Start Dictation：这是 Mac 里专有的菜单，主要是启用听写功能，用“说的”来写案例，感兴趣的可以试试。

接下来的几个 Cells 和 Rows 的操作，大部分同学应该都用过 Excel 吧，后面我们写案例的地方和 Excel 的表格很类似，所以 Cells 和 Rows 就是对单元格和行进行操作。

3. Tools 工具

Windows 平台的 Tools 菜单，如图 2-4-6 所示。Mac 平台的 Tools 菜单，如图 2-4-7 所示。

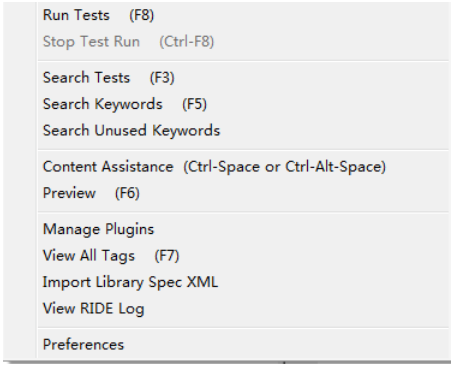

图 2-4-6

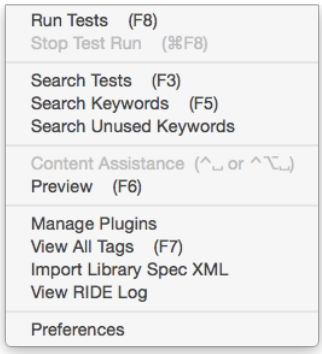

图 2-4-7

Tools 菜单下的主要菜单项，见表 2-4-3 所示。

表 2-4-3

Run Tests	运行测试案例
Stop Test Run	停止案例运行
Search Tests	搜索测试案例
Search Keywords	搜索关键字
Search Unused Keywords	搜索未使用的关键字
Content Assistance	内容助手
Preview	预览
Manage Plugins	管理插件
View All Tags	查看所有Tag
Import Library Spec XML	导入测试库描述文件xml
View RIDE Log	查看RIDE日志
Preferences	参数配置

Run Tests 和 Stop Test Run 两个菜单项对应图标为 ，简单理解就是一个开始运行案例，一个停止运行案例。

Search Tests：用来搜索测试案例，可以通过名称搜索或者 Tag 标签搜索，如图 2-4-8 所示。

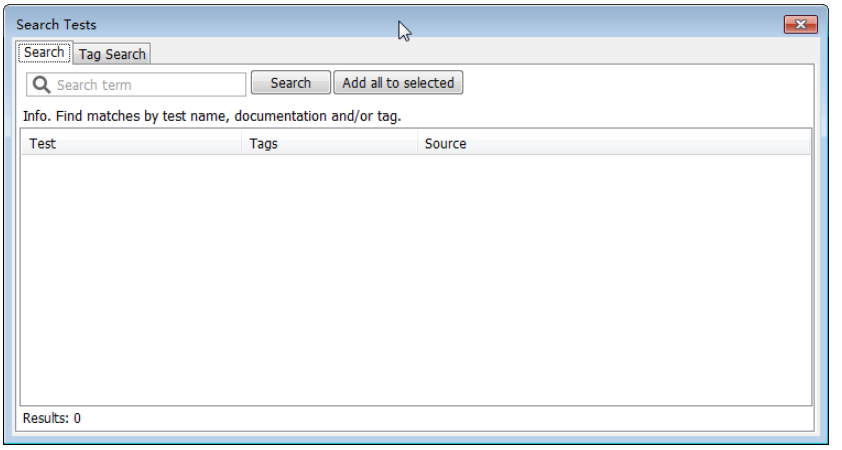

图 2-4-8

Search Keywords：另外笔者认为比较常用的一个是 Search Keywords（快捷键 F5）。对于我们在写案例时，可以方便地查找测试库的关键字以及其参数和样例等。如图 2-4-9 所示。

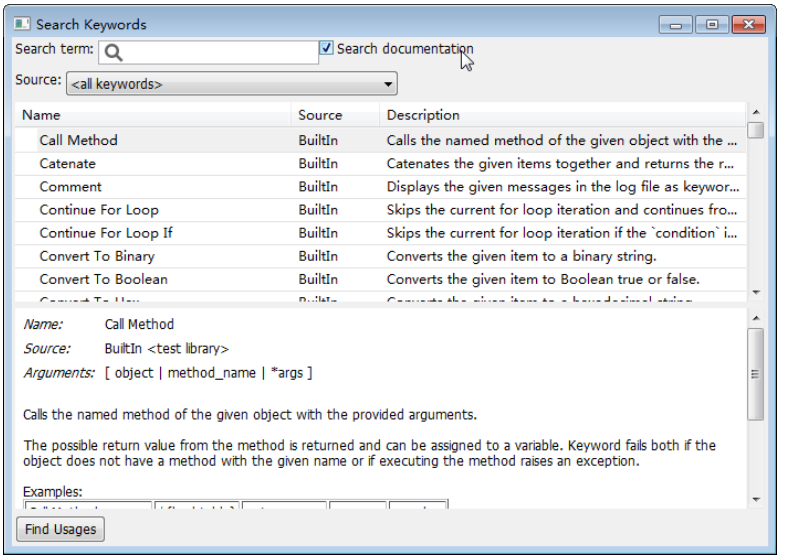

图 2-4-9

Search Unused Keywords：这里有个新增的功能 Search unused Keywords。查找没有用过的关键字，关键字多了以后可能会有些没有用过的，这个可以比较方便地查找，如图 2-4-10 所示。

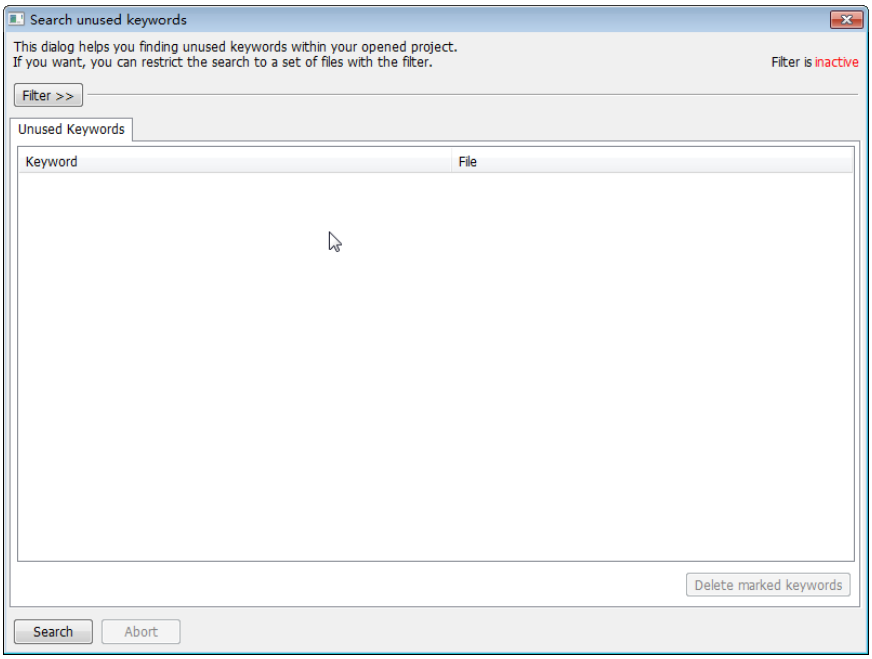

图 2-4-10

Content Assistance：这个功能是内容助手，用来做脚本补全的，很有用但是不太方便用，主要是快捷键的冲突问题。因为“Ctrl+空格”组合键，和中英文切换操作冲突了，而“Ctrl+Alt+空格”组合键又按着不太习惯。笔者的做法是把输入法的快捷方式“Ctrl+空格”组合键改成其他的组合键，这样就可以使用了，如图 2-4-11 所示。

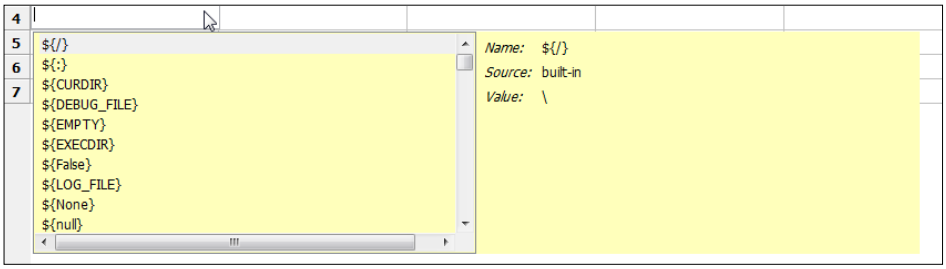

图 2-4-11

Preview：预览测试套件的文件内容，可以选择 HTML 或者 txt 文本格式，查看或打印。如图 2-4-12 所示。

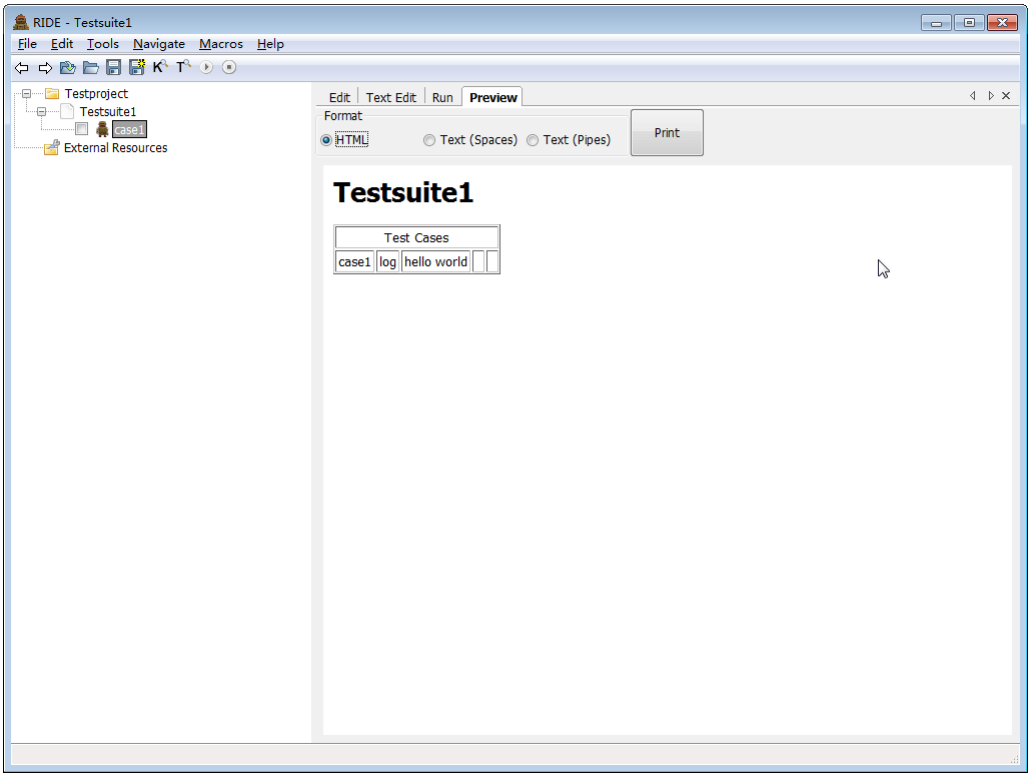

图 2-4-12

Manage Plugins：管理 RIDE 的插件，像关键字搜索、预览等，都是以插件的形式集成进来的。不建议初学者随意修改这里的插件状态。如果熟悉了 RIDE 的内部逻辑，以后自己开发个插件集成进来也是可以的。如图 2-4-13 所示。

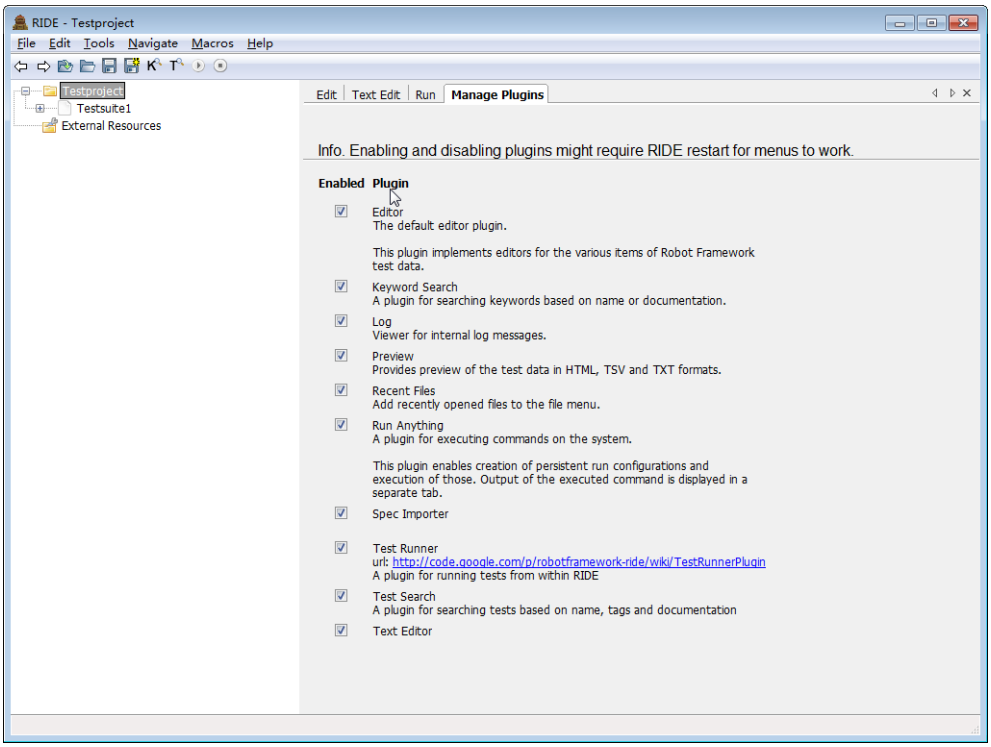

图 2-4-13

View All Tags：查看所有的 tags 标记，后面会讲到案例的 tags 标记。这里就是方便大家来查看自己标记的案例，也可以进行相应的搜索，如图 2-4-14 所示。

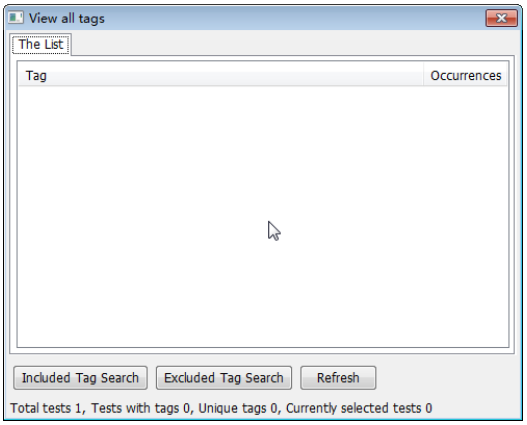

图 2-4-14

Import Library Spec XML：这个功能是新出来的，主要作用是方便那些没有安装功能的测试库 TestLibrary，通常是单独的 py 文件的或者 JavaLibrary，用命令生成出这个测试库的 xml 文档，在导入后就把他变成类似于安装后的测试库一样来使用。不过我们目前基本用不上，单独 py 文件的测试库其实只要放到 site-packages 目录就可以了，Java 的 Library 我们本书暂不涉及。

View RIDE Log：这个功能作用非常大。特别是在测试库名字写错了的时候，可以通过这个 log 看到错误信息，帮助大家找问题，如图 2-4-15 所示。

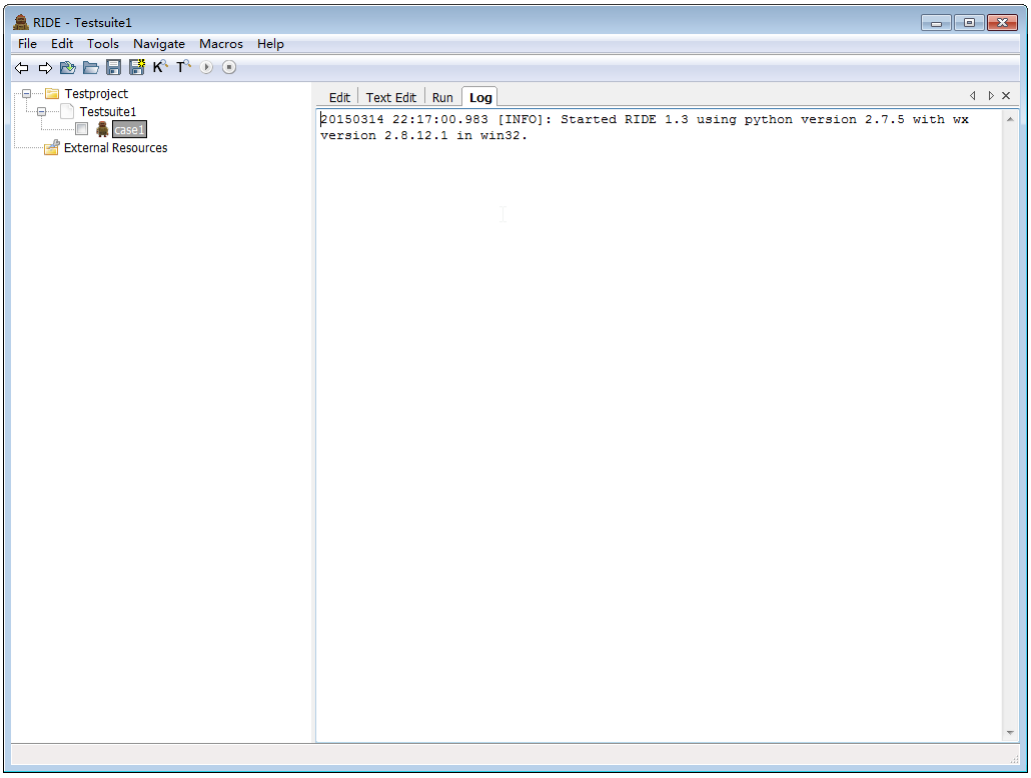

图 2-4-15

Preferences 是偏好设置，这里可以进行很多基础设置，这里介绍几个常用的。

（1）Saving：这里可以设置默认的文件格式、文本格式分隔符、行分隔符、单元格之间 4 个空格分隔。这里一般不建议大家修改，默认文件格式可以修改成你自己喜欢的格式。如图 2-4-16 所示。

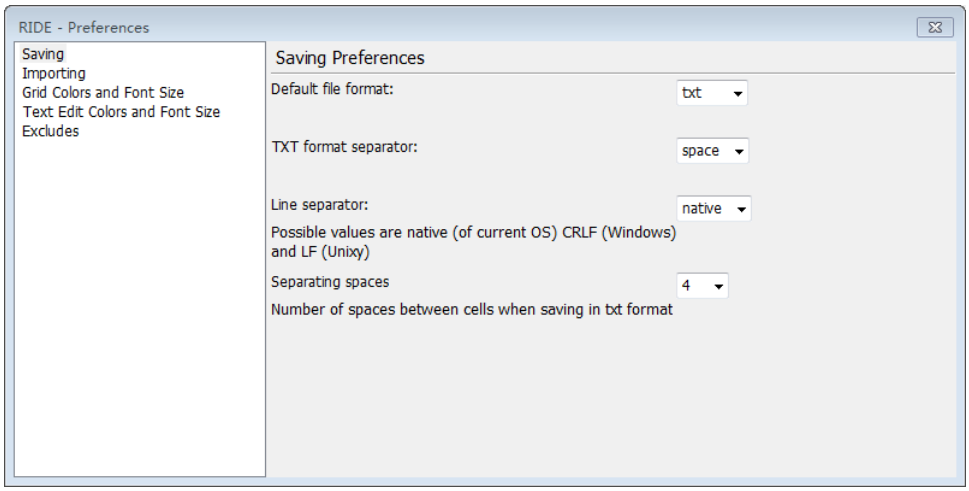

图 2-4-16

（2）Importing：可以设置自动导入哪些 Library 库，PythonPATH 路径用于搜索 Library 库，以及 Library xml 的目录。这些基本上都不需要做修改，如图 2-4-17 所示。

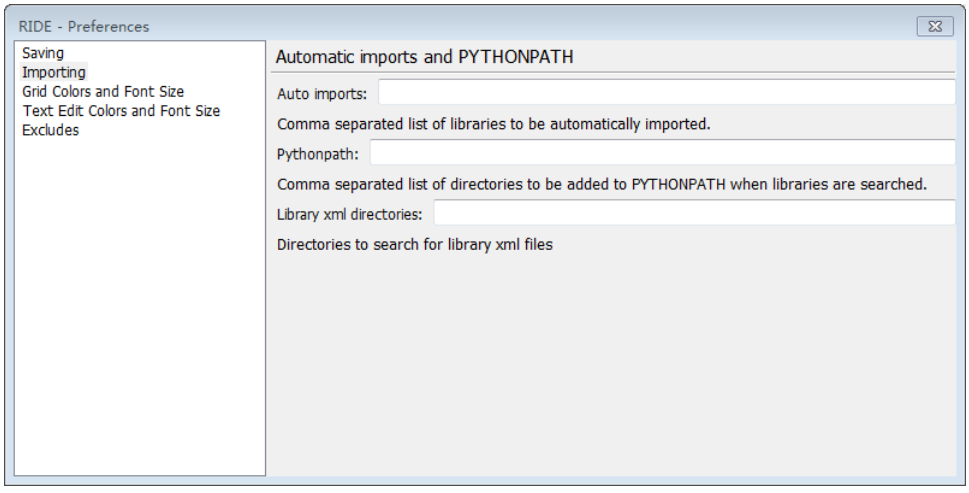

图 2-4-17

（3）Grid Colors and Font Size：用于设置写案例的表格里的字体大小和颜色，如果你觉得默认的字体比较小，可以改大一些。颜色一般不建议修改，默认就可以了，如图 2-4-18 所示。

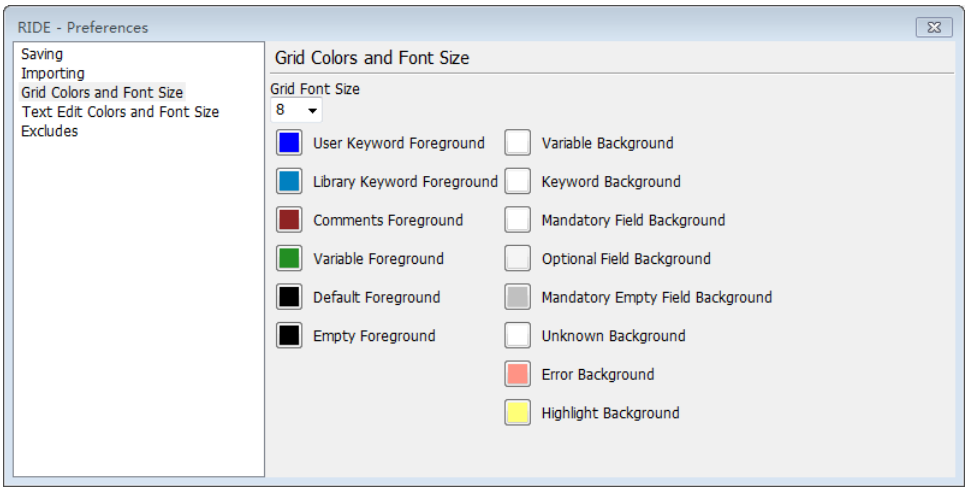

图 2-4-18

（4）Text Edit Colors and Font Size：用于设置写案例的文本编辑模式里的字体大小和颜色，不过一般很少去那个页面进行编辑，如图 2-4-19 所示。

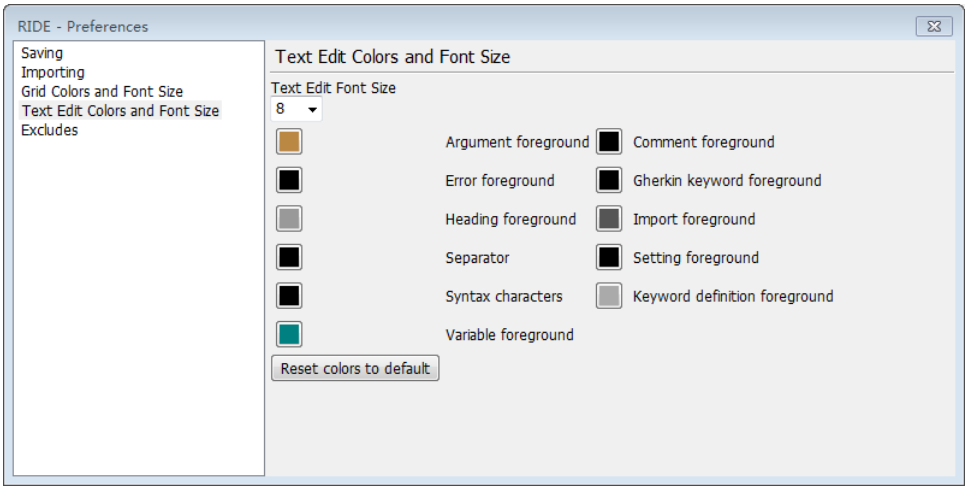

图 2-4-19

（5）Excludes：里面就一个文本框设置 Excludes，根本不知道是什么。幸好有个“Need help？”选项，可以单击该选项寻求指导和帮助，如图 2-4-20 所示。

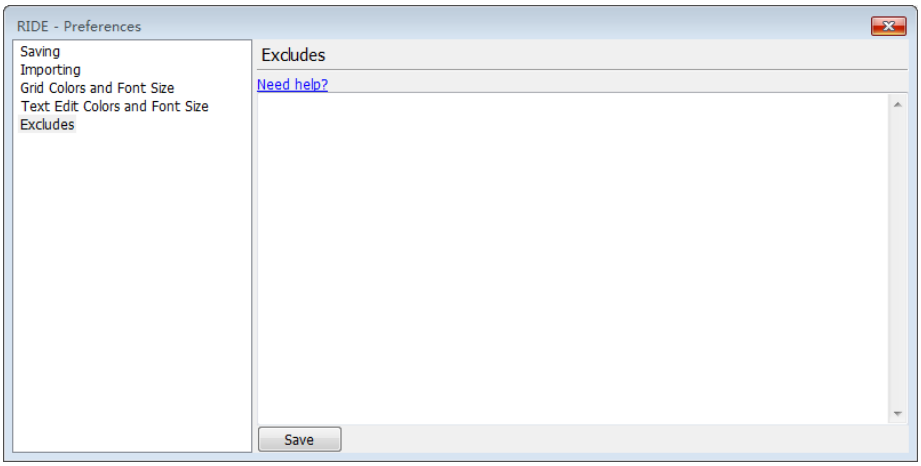

图 2-4-20

在弹出的对话框中有具体的介绍，如图 2-4-21 所示。。可以在文本框里输入一个你想要排除的路径，每行一个，当你保存后重新加载工程才会生效。通常用于在你的工程目录内，还有其他的不是测试套件的目录，可以在这里进行设置，那样 RIDE 就不会去识别该目录下面的文件了，大家可以自己试一下。

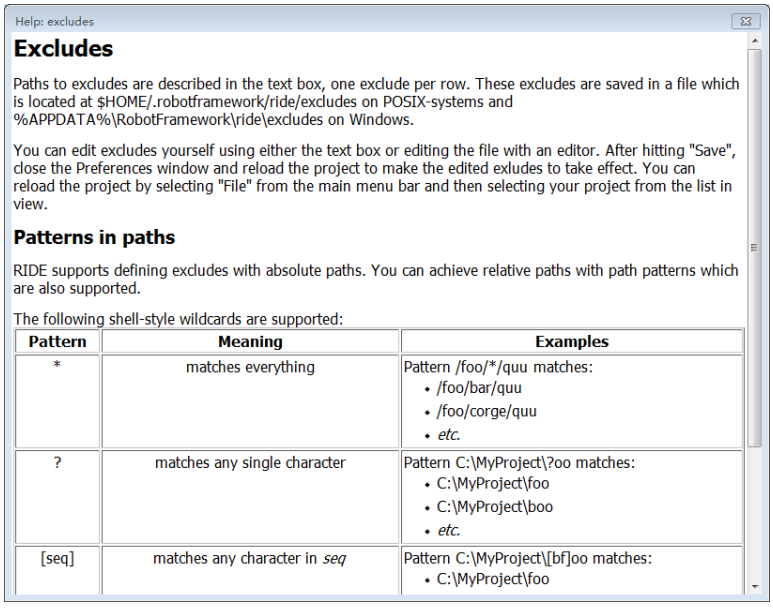

图 2-4-21

4. Navigate 导航

Windows 平台的 Navigate 菜单，如图 2-4-22 所示。Mac 平台的 Navigate 菜单，如图 2-4-23 所示。

图 2-4-22

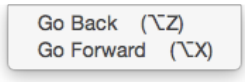

图 2-4-23

Navigate 菜单下的菜单项，见表 2-4-4。

表 2-4-4

Go Back	后退
Go Forward	前进

后退和前进，可以方便地在你访问过的页面进行跳转，对应图标为⇦ ⇨。

比如你先点了 testsuite1 看他的页面，然后又点了 case1，此时就可以用后退和前进切换两个不同的页面。

5.Windows 窗口（Mac）

对比 Mac 和 Windows 操作系统，在 Mac 上多了一个 Windows 的菜单。这是 Mac 操作系统自带的，每个软件都会有，如图 2-4-24 所示。

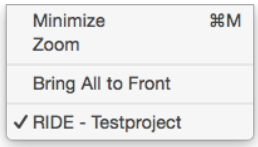

图 2-4-24

Windows 菜单下的菜单项，见表 2-4-5。

表 2-4-5

Minimize	最小化
Zoom	最大化
Bring All to Front	把所有窗口前置

RIDE-Testproject 前面打勾，是标识当前 RIDE 是打开了哪个工程，如果你开了多个 RIDE，这里就会有多个项目，可以根据自己需要进行切换。

6. Macros 宏命令

看过笔者博客的人应该知道，早期版本这个菜单叫 Run，当时笔者觉得这个名字并不合适。现在官方将它改成 Macros 宏命令后，这回比较合适了。

Windows 平台的 Macros 菜单，如图 2-4-25 所示。Mac 平台的 Macros 菜单，如图 2-4-26 所示。

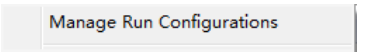

图 2-4-25

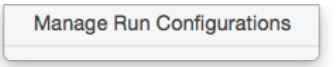

图 2-4-26

Macros 的菜单项就这一个 Manage Run Configurations，即管理运行配置。

这个设置平时用的比较少，简单研究了一下，就是可以自己写好一些命令行的语句，可以直接通过菜单来运行。主要是针对通过命令行方式运行的一些命令，可以自定义菜单，菜单里也给了一些例子。

比如笔者简单配置了 2 个，如图 2-4-27 所示。

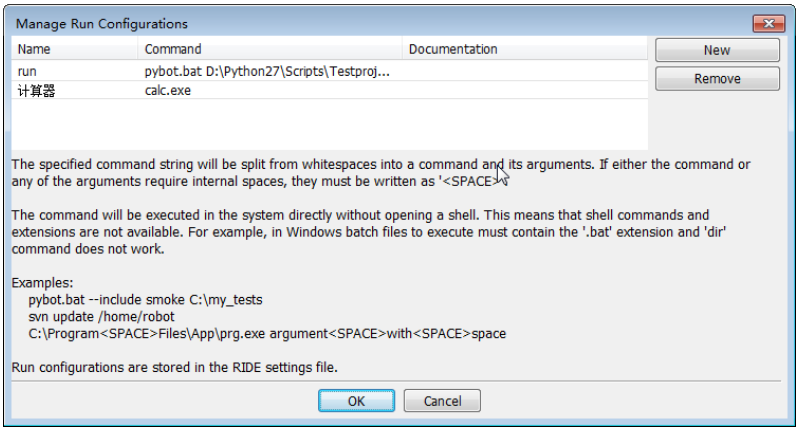

图 2-4-27

再单击 Macros 的菜单，就多了这 2 个了，可以直接单击运行。如图 2-4-28 所示。

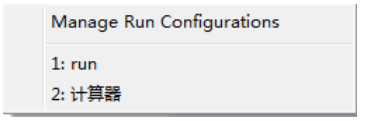

图 2-4-28

选择第二个运行一下看看，它直接在工作区增加了一个 tab 页，运行结果也会显示在上面。所以，如果你平时有什么需要使用的，可以加到菜单里。如图 2-4-29 所示。

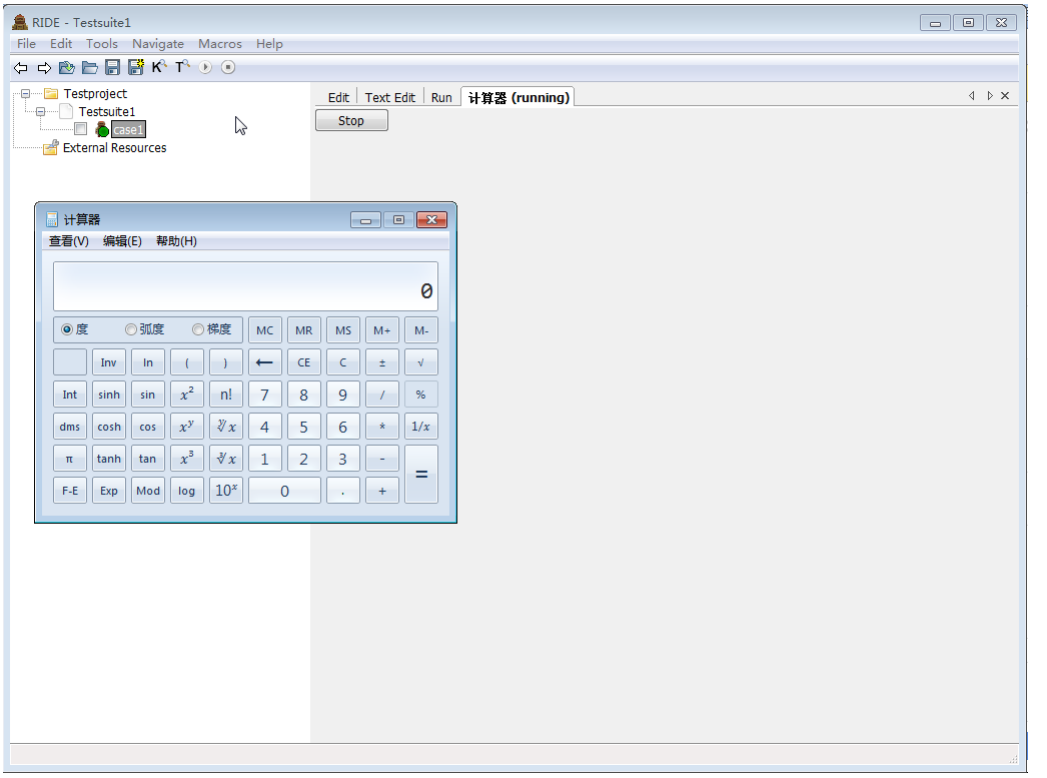

图 2-4-29

Mac 平台上也是类似的，这里就不再演示了。

7. Help 帮助

Windows 平台的 Help 菜单，如图 2-4-30 所示。Mac 平台的 Help 菜单，如图 2-4-31 所示。

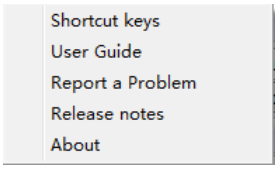

图 2-4-30

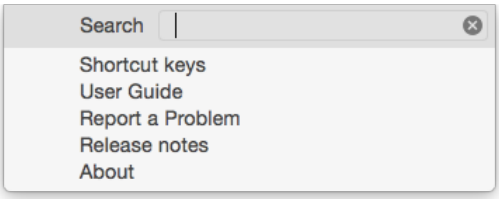

图 2-4-31

Help 菜单下的菜单项，见表 2-4-5。

表 2-4-5

Shortcut keys	快捷键列表
User Guide	用户指南
Report a Problem	报告问题
Release notes	更新日志
About	关于

- Shortcut keys：快捷键列表。可以让你查找所有的快捷键，以后经常使用快捷键，可以提高一定的效率；
- User Guide：用户指南其实是 Robot Framework 的指南。由于 google code 关闭，用户指南已经迁移到了其他网址（http://robotframework.org/robotframework/）；
- Report a Problem：如果你要报告 RIDE 问题，最好到 github 上去提交问题。不过这里的菜单项还是指向了 google code 的页面，估计以后的版本会更新过来；
- Release notes：更新日志，可以看一下每个版本都修改了哪些问题或加了哪些功能；
- About：关于。

在 Mac 平台多了一个 Search 搜索，是用来搜索菜单项的，这其实是 Mac 平台自带的。

特别当你想要找某个菜单项时，当你把鼠标移动到搜索结果上，它还会给你指出菜单在哪里，如图 2-4-32 所示。

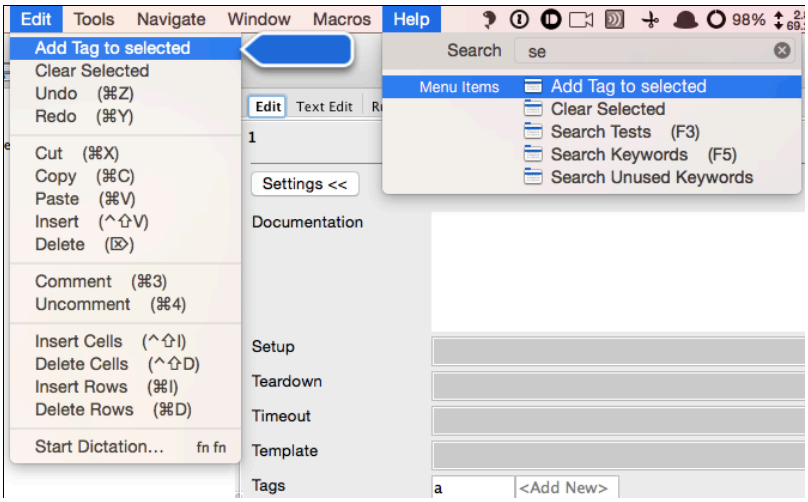

图 2-4-32

2.4.3　工作区

工作区里的内容其实很多，工程、测试套件、测试案例、资源、用户关键字等，都会有 Edit 界面，但是又稍有不同，先把通用的介绍给大家，其他的细节将分散在各个模块详细介绍。

1. Edit 界面

（1）测试套件的 Edit 界面

选择测试套件后，看右侧工作区的 Edit 界面，如图 2-4-33 所示。资源的工作区 Edit 与测试套件的基本相同，除了一点，就是它没有最后一个元数据定义的部分。

第一行的“Source”列出了这个 TestSuite1 的路径。接着有个“Settings”按钮，现在是隐藏了内容的，单击一下会显示出具体的 Setting 内容，在后面的章节里，每个不同的模块会单独介绍它的 Settings 内容。

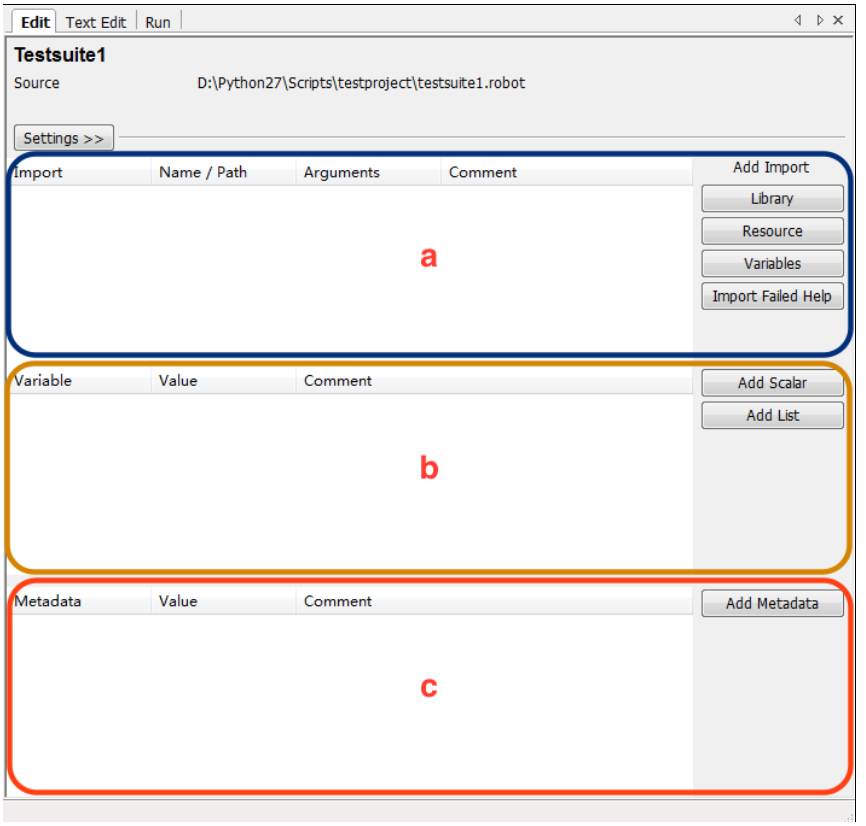

图 2-4-33

再往下可以大体分成 3 个部分：

1）Add Import 为添加导入文件。各个按钮说明，见表 2-4-6。

表 2-4-6

Library	加载测试库，默认是“PYTHON安装目录”\Lib\site-packages里的测试库
Resource	加载资源（主要是工程相关的资源文件）
Variables	加载变量文件
Import Failed Help	导入失败的帮助

2）Add Variable 为添加变量。各个按钮说明，见表 2-4-7。

表 2-4-7

Add Scalar	定义变量
Add List	定义列表型变量

3）定义元数据

Add Metadata 为定义元数据（笔者是直接翻译的）。这个是新增加的部分，大概看了一下作用，是在 Report 和 Log 里显示定义好的内容，格式和 document 一样。

（2）测试案例的 Edit 界面

单击一个测试案例，可以看到它的工作区是这样的，如图 2-4-34 所示。用户关键字的工作区 Edit 和测试案例的也是基本一样的，只是用户关键字的界面多了一个 Find Usages。

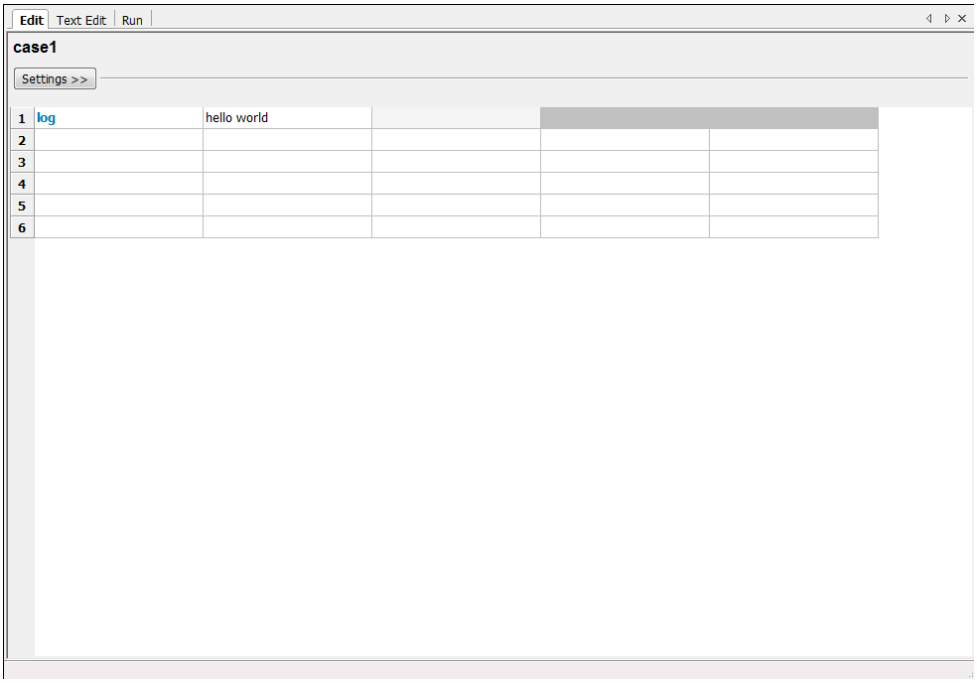

图 2-4-34

2. Text Edit 文本编辑界面

这里就是显示我们那些文件里的真实内容，大家可以切换一下 Edit 和 Text Edit，就会知道用 RIDE 还是很方便的，否则要把案例写成 Text Edit 界面里看到的这样，如图 2-4-35

所示。现在案例内容比较少，可能看不出来，等案例多了以后，再看一下这个页面。

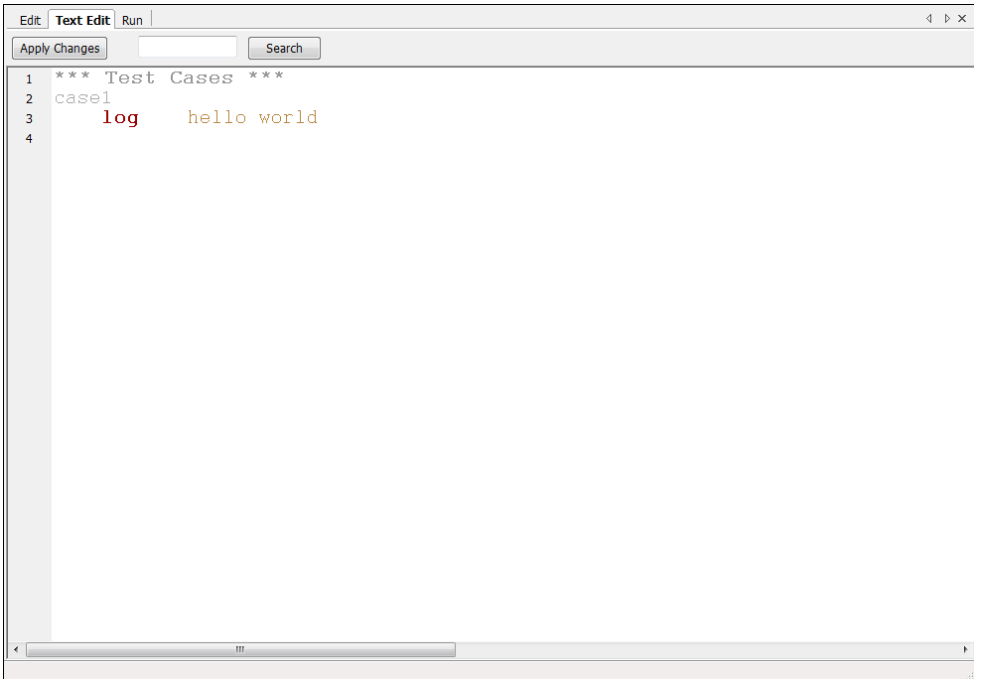

图 2-4-35

3. Run 运行界面

这个界面里有一些运行时的参数设置、日志显示等，如图 2-4-36 所示。在后面会有专门的一个章节来介绍相关内容。

工程文件区就不单独介绍了，因为后续的各个模块里都会有这里的内容。这一节把 RIDE 的一些内容大体介绍了一下，先让大家有个初步的印象和了解，后续还有更多的细节都会整合到具体模块的细节介绍中。

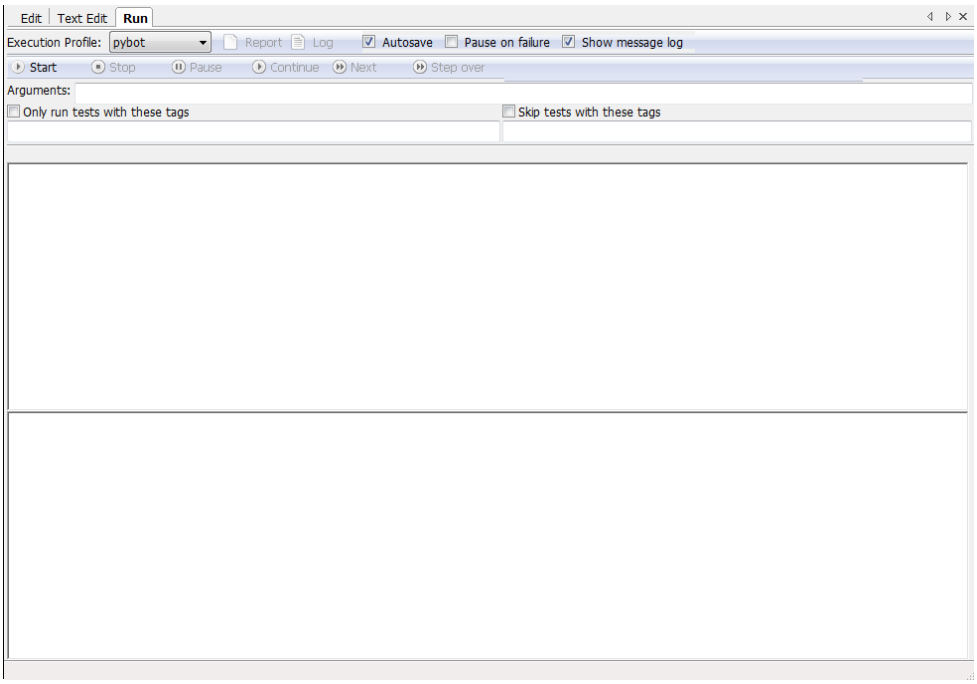

图 2-4-36

2.5　工程、测试套件、测试案例

2.5.1　Project 工程

1. 新建工程

还记得在前面第一个例子里，如何新建工程的么？不记得也不要紧，我们再来一遍，单击“File→New Project”命令，如图 2-5-1 所示。

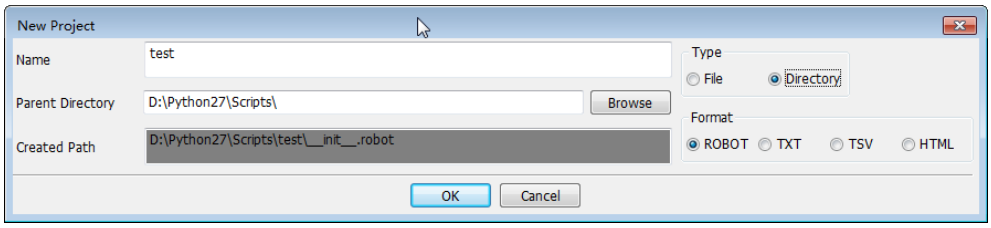

图 2-5-1

界面上的主要内容说明，见表 2-5-1。

表 2-5-1

Name	工程命名
Parent Directory	上级目录，工程会创建在这个目录下
Created Path	创建工程后的完整路径
Type	工程类型
Directory	文件格式

建议创建工程时看清楚 Parent Directory 这个目录，因为它默认是前一次工程的目录。

Type 分为文件和目录两种，区别要看你的工程来定。如果内容很简单，File 类型的就可以；如果内容较多，从方便管理的角度来说，选 Directory 的比较合适。这里我们选择目录。

Format 文件格式，分为 ROBOT、TXT、TSV、HTML 这几种格式，默认 ROBOT 格式就很不错，也可以考虑 TXT 格式，便于维护和版本管理。后续的案例，我们都是以 ROBOT 为默认选择。

单击“OK”按钮，工程创建成功，会在界面左侧看到 Test 的目录图标，如图 2-5-2 所示。

2. 快捷菜单

先看一下在目录型的 Project（目录型的 Test Suite）下可以做些什么，在 Project 名字上单击鼠标右键，弹出快捷菜单，如图 2-5-3 所示。

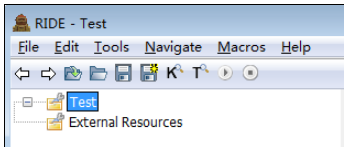

图 2-5-2

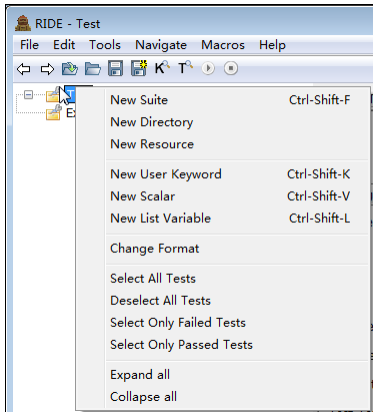

图 2-5-3

快捷菜单菜单项，见表 2-5-2。

表 2-5-2

New Suite	新建测试套件
New Directory	新建目录测试套件
New Resource	新建资源
New User Keyword	新建用户关键字
New Scalar	新建Scalar变量
New List Variable	新建List变量
Change Format	修改格式
Select All Tests	选择全部测试案例
Deselect All Tests	取消选择全部测试案例
Select Only Failed Tests	只选择失败的测试案例
Select Only Passed Tests	只选择成功的测试案例
Expand all	展开全部目录
Collapse all	折叠全部目录

每个菜单项的功能都比较直观，就不一一解释了，重点说一下其中的 2 个。

- New Directory：创建工程或者测试套件时，还可以选择一下是文件还是目录，这个菜单项直接默认就是目录了。
- Change Format：之前设定了工程或测试套件的格式，这里可以进行修改。

后面的快捷菜单也有很多重复的，笔者只介绍不同的菜单项。

3. Settings 设置项

前面看了工作区的 Edit 界面，大体上测试套件的外观基本上是一致的，细节有些不同。这里就来看一下目录型的 Project（同时也是目录型的 Test Suite 的，原因在后续内容中会说明）的 Settings 设置项，如图 2-5-4 所示。

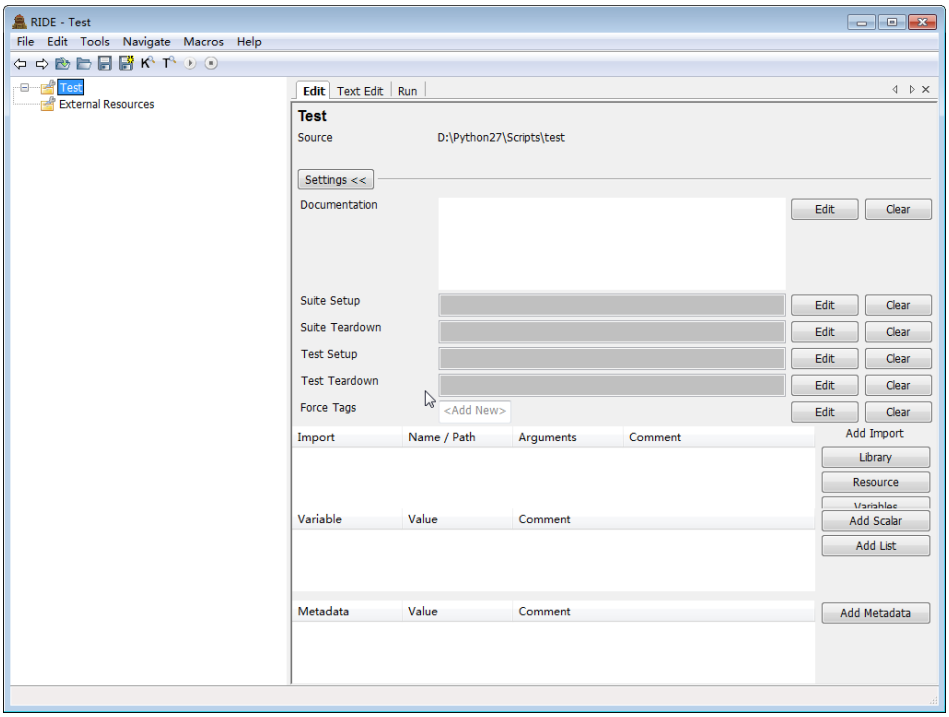

图 2-5-4

设置项菜单项，见表 2-5-3。

表 2-5-3

Documentation	说明文档
Suite Setup	测试套初始化
Suite Teardown	测试套结束
Test Setup	测试案例初始化
Test Teardown	测试案例结束
Force Tags	强制标记

Documentation 可以写一下当前的工程的说明，如果想加粗，可以用“*要加粗的内容*”设置加粗；如果想斜体，可以用“_要斜体的内容_”设置斜体。

Setup 和 Teardown，如果有用过 JUnit 的读者，应该比较熟悉这 2 个菜单项，简单地说就是初始化和结束。用一个不太雅观的例子，比如你要上厕所，但是要先拿纸，完事儿

后要擦屁股，所以对于上厕所这件事儿来说，拿纸就是 Setup，擦屁股就是 Teardown，为啥用这个例子，因为笔者觉得 Teardown 就是在做很多“擦屁股”的事情，特别是在后面的 Web 测试里。

熟悉了 Setup 和 Teardown，那么就比较好理解 Suite 的 Setup 和 Teardown，这两个菜单项指的是测试套件的初始化和结束，Test 的 Setup 和 Teardown 指的就是测试案例的初始化和结束。

假设 Suite1 下面有 Test1 和 Test2，假设它们都设置了 Setup 和 Teardown，那么它们的执行顺序是怎样的呢？对于 JUnit 里面的顺序是“Setup→执行测试→Teardown”。那么对应到我们这个例子的顺序就是“Suite1-Setup→Test1-Setup→Test1-执行→Test1-Teardown→Test2-Setup→Test2-执行→Test2-Teardown→Suite1-Teardown”。

Force Tags 为强制标记，这里的强制标记就是给当前测试套件下的每个测试案例都加上 Tags，而且不能在案例上删除这个 Tags，只能到设置的地方删除。

直接单击每一项的文本框，或者后面的“Edit”按钮，可以打开编辑界面，如果想删除的话，可以单击“Clear”按钮。

2.5.2　Test Suite 测试套件

1. 新建测试套件

在 Test 上单击鼠标右键，弹出快捷菜单，如图 2-5-5 所示。

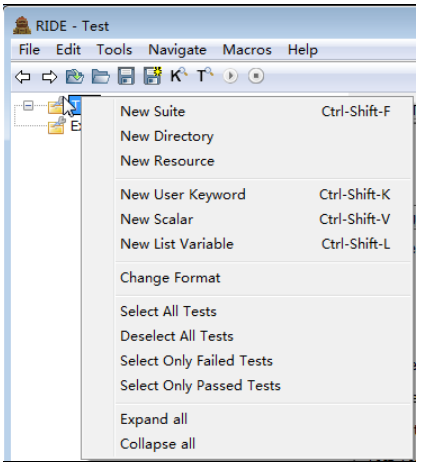

图 2-5-5

选择“New Suite”命令，建立一个测试套件，如图 2-5-6 所示。

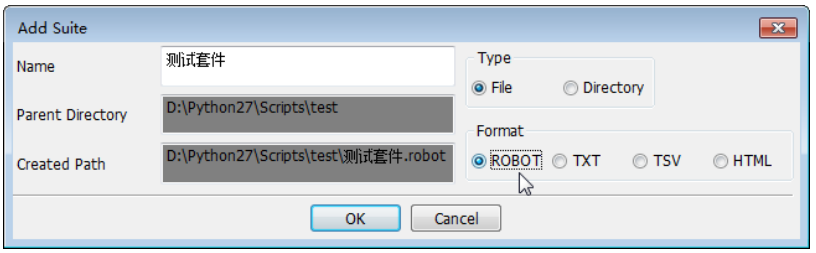

图 2-5-6

看到这个图，比较一下和前面 New Project 的图有区别么？

其实，从本质上说，Project 和 Test Suite 是一样的。如果硬要区分一下的话，笔者的意见是最顶层的 Test Suite 就是 Project。对于初学者来说，完全可以把它们当作同一个东西来理解。这里选择“File”单选按钮（File 类型的），单击“OK”按钮，如图 2-5-7 所示。

2. 快捷菜单

接下来看看文件型的 Test Suite（文件型的 Project）能做什么。在“测试套件”上单击鼠标右键，弹出快捷菜单，如图 2-5-8 所示。

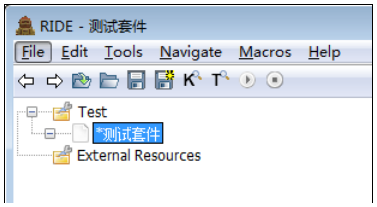

图 2-5-7

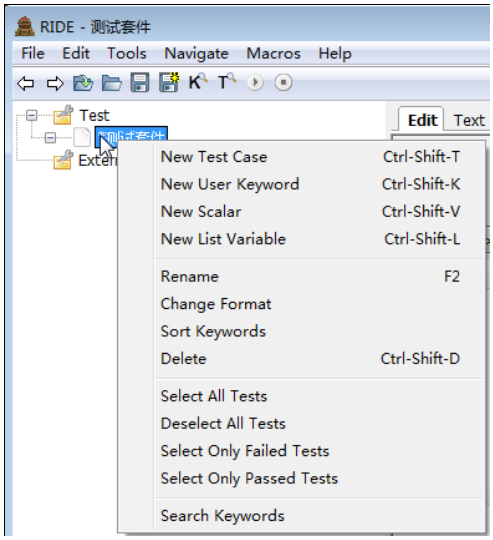

图 2-5-8

有很多菜单项和之前工程的菜单项是一样的，这里就跳过重复的，说一下不同的。

首先是原来的 New Suite、New Directory、New Resource 没有了，这里换成了 New Test Case（新增测试案例）。

Rename 和 Delete 就不多说了。Sort Keywords 会对当前 Suite 下面的关键字进行重新排列整理，Search Keywords 和主菜单里的搜索关键字是一样的。

3. Settings 设置项

看一下 File 类型的 Test Suite 的 Settings 设置项，如图 2-5-9 所示。

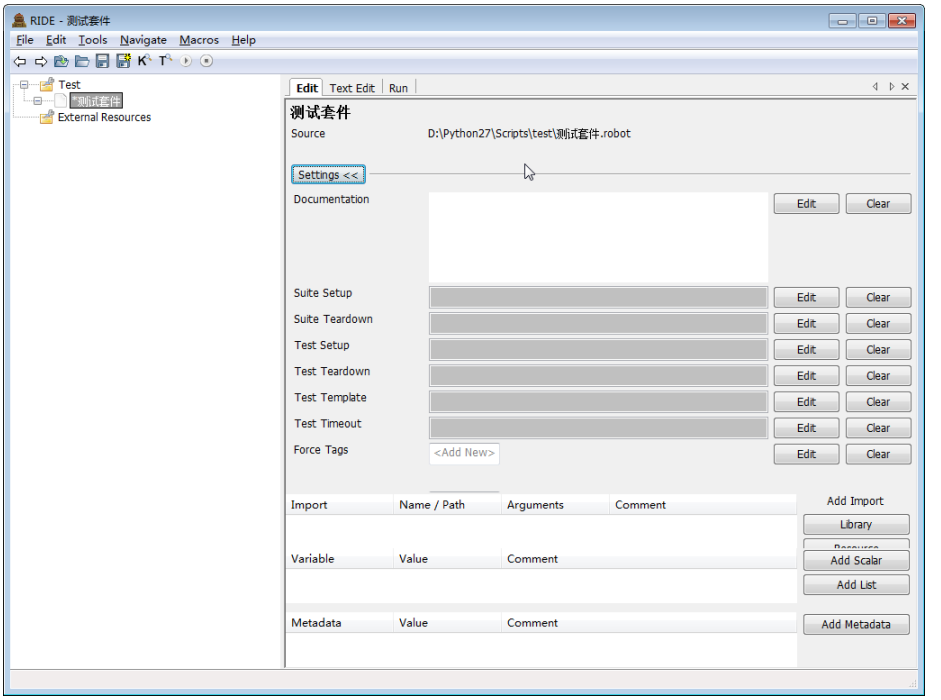

图 2-5-9

这里和前面工程里的大体上差不多，多了几项都是和 Test Case 相关的，这是因为我们的 Test Case 只能放在 File 类型的 Test Suite 下面。

Test Template 为测试案例的模板，后面讲 Web 测试的 Selenium2Library 会看到这个的用法，但是笔者不推荐大家使用。

Test Timeout 为测试案例的超时时间，如果某个案例执行超时就结束该案例运行。

2.5.3 Test Case 测试案例

1. 新建测试案例

在“测试套件”上单击鼠标右键，弹出快捷菜单，如图 2-5-10 所示。

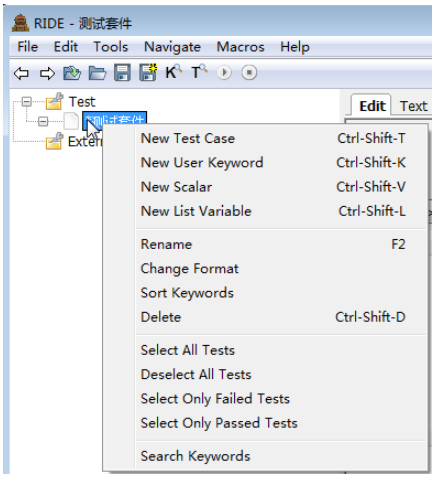

图 2-5-10

单击“New Test Case”命令，输入案例名称为 case，如图 2-5-11 所示。

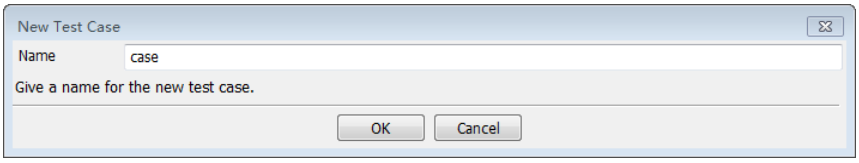

图 2-5-11

单击“OK”按钮，然后显示的界面如图 2-5-12 所示。

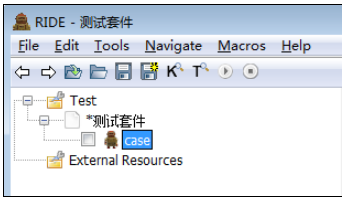

图 2-5-12

看见测试套件前面多了个“*”么？这表示他是有了新的修改，还没有保存，所以先

保存一下。

2. 快捷菜单

这里看一下在 Test Case 下能做些什么，在“case”上单击鼠标右键，如图 2-5-13 所示。

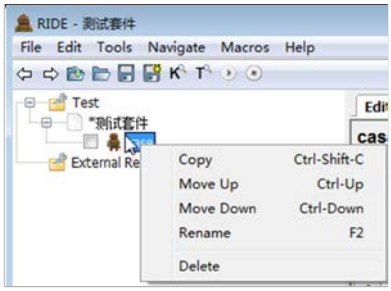

图 2-5-13

这里就只有几项编辑选项了。

Copy 是复制当前案例，产生一个相同的新案例。

Move Up 和 Move Down 是当案例多了以后，可以调整案例的顺序。

3. Settings 设置项

看一下 Test Case 的 Settings 设置项，如图 2-5-14 所示。

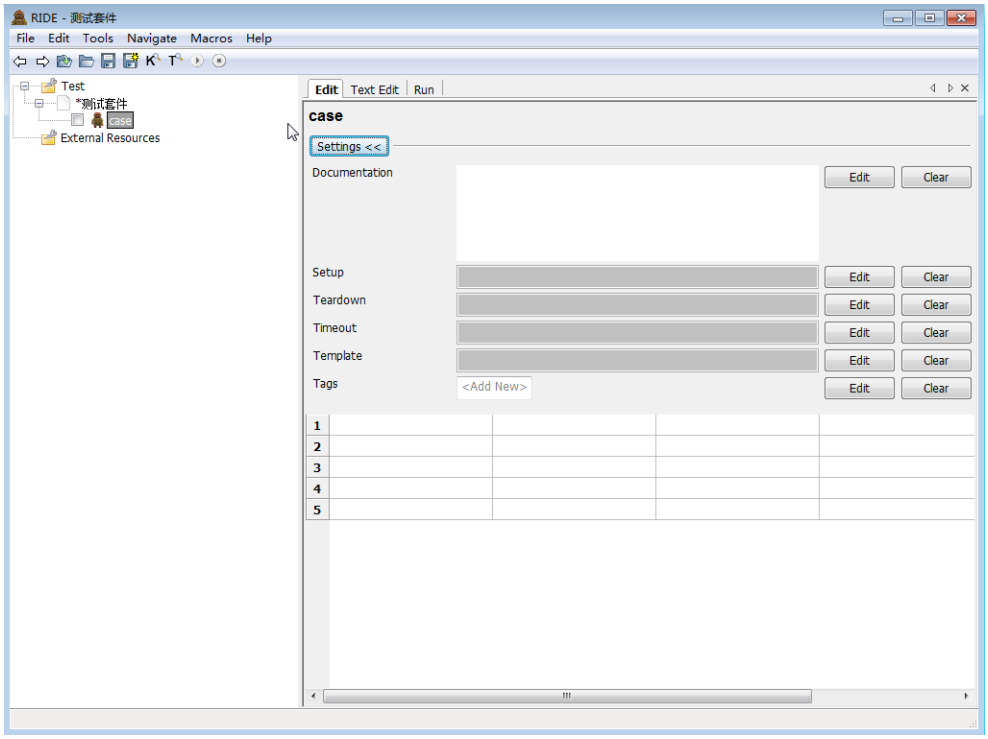

图 2-5-14

一共 6 个设置，Documentation 就不用说了，中间的 4 个和 File 型的 Test Suite 的 Settings

的那几项 Test 的设置一样，就是名字上把 Test 去掉。而 Tags 去掉了 Force，因为 Case 已经是最底层了，这里的 Tags 就只是针对案例自身进行设置。

在后面的章节，会看一下用户关键字和测试案例有什么差异。

2.5.4 三者关系

请注意是三者关系，不是另一个关系。这三者主要指的是前面 3 节的工程、测试套、测试案例。这里用一个关系图来说明一下，如图 2-5-15 所示。

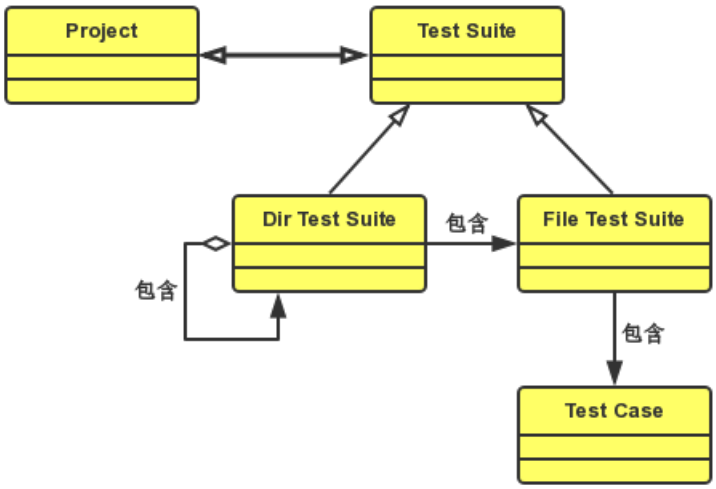

图 2-5-15

在图 2-5-15 中，Project 和最顶层的 Test Suite 是相通的，可以是 File 文件类型，也可以是 Directory 目录类型。

作为一个目录型的 Dir Test Suite 来说，它们下面可以包含 File Test Suite 或者 Dir Test Suite，Dir 的层级可以很深。但最终要用的 Test Case 只能在 File Test Suite 下面增加。可以在 Dir Test Suite 上单击鼠标右键，肯定看不到“New Test Case”选项。

大家可以自己创建一些复杂的目录结构，来体验一下 Project、Test Suite 和 Test Case 的关系。比如在刚才详细介绍的过程里，又创建了一个和第一个案例平级的 Project，然后用 RIDE 的 File 下面的 Open Directory 打开这个 Project 的上级目录，如图 2-5-16 所示。

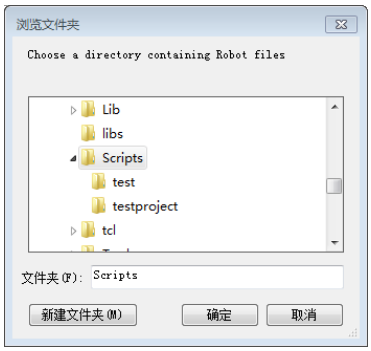

图 2-5-16

你会发现此时的 Project 是这个上级目录了，同时原先的 2 个 Project 都成了 Dir Test Suite 了，如图 2-5-17 所示。

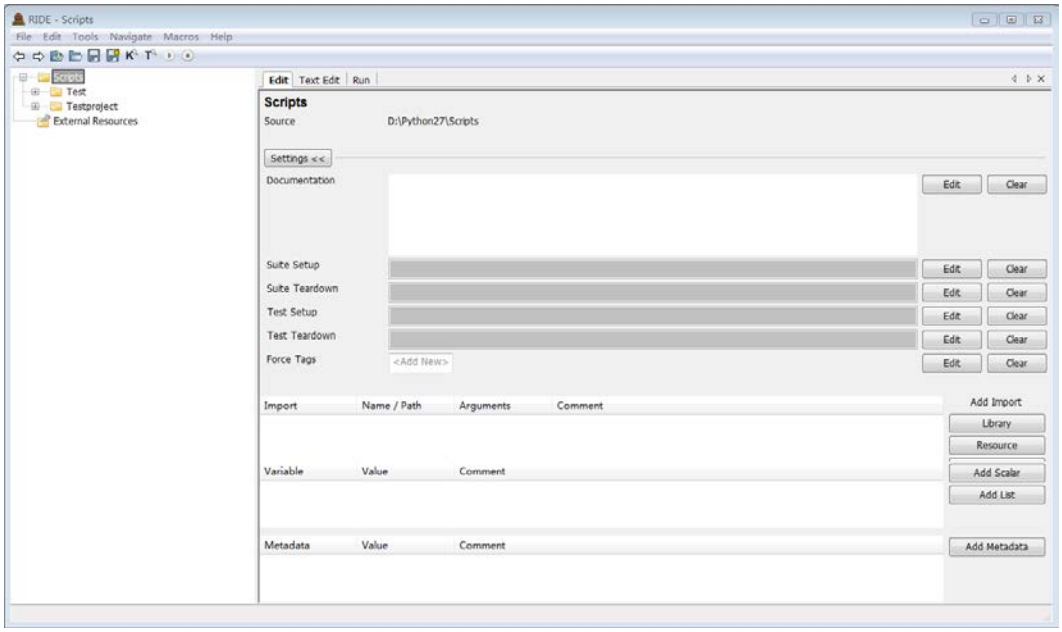

图 2-5-17

2.6　测试库

如果说 Robot Framework 是一棵大树，测试库（TestLibrary）就是繁茂的枝叶，正是

有了众多的测试库，才能让我们更方便地选择合适的测试库来进行自动化测试。

在 Test Suite 或者 Resource 里都可以增加 Library 测试库，通常的建议是在 Resource 里添加 Library 测试库，在 Test Suite 里添加 Resource，形成一个串联，这种模式在 Web 测试中会讲解。

选中测试套件，在界面右侧的工作区单击“Library”按钮，弹出“Library”对话框，如图 2-6-1 所示。

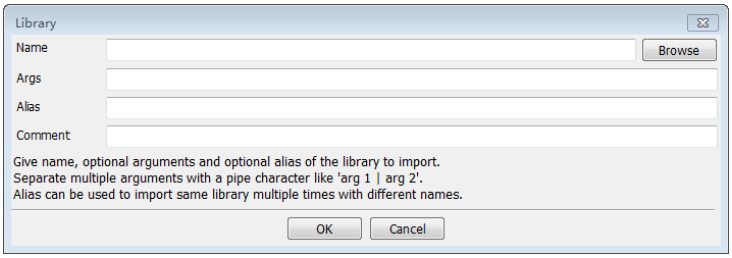

图 2-6-1

在“Name”文本框中输入测试库的名字，可以是测试库的名字，也可以通过单击“Browse”按钮，在弹出的对话框中选择测试库的文件。

在“Args”文本框中输入测试库的参数，一般都可以默认不加参数。如果想要加参数，需要先了解这个测试库都有什么参数。

在“Alias”文本框中可以输入测试库的别名，作用是你可以通过不同的别名来导入同一个 Library。

添加测试库时最少需要写个测试库的名字，在 Robot Framework 中内置了一些非常好用的测试库，后面有专门的章节介绍它们。其中有一个测试库叫 String，这里演示一下如果写错测试库名字会怎样，笔者故意输入了 Strings 这个名字，如图 2-6-2 所示。

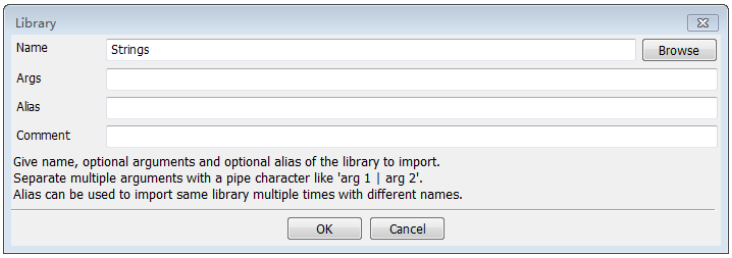

图 2-6-2

单击“OK”按钮，弹出的界面如图 2-6-3 所示。

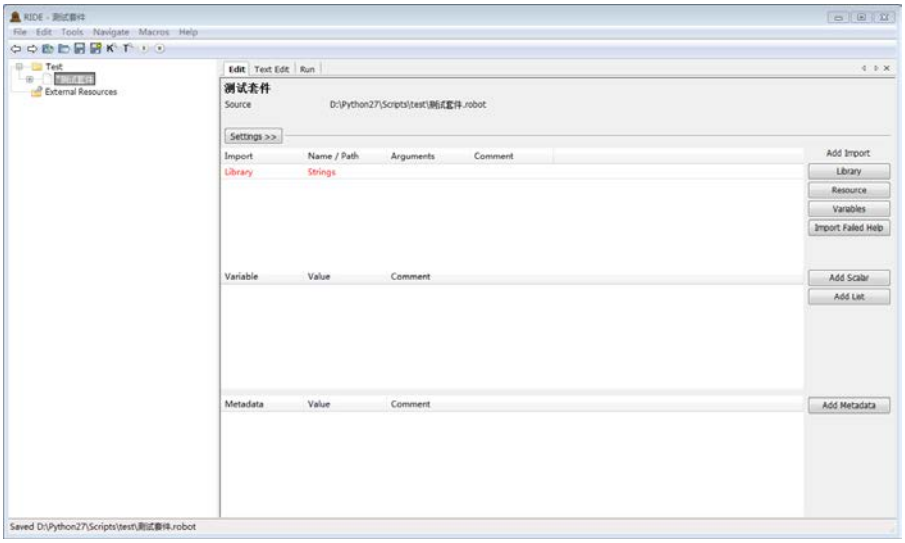

图 2-6-3

这里会看到 Import 里的 Library 是红色的 Strings，这说明 Import Failed，在 Add Import 下面有个失败说明，它只是告诉你去哪里看。前面介绍菜单时已经介绍过了，在 Tools 下的 View RIDE Log 可以查看日志，操作一下后会看到弹出的界面，如图 2-6-4 所示。

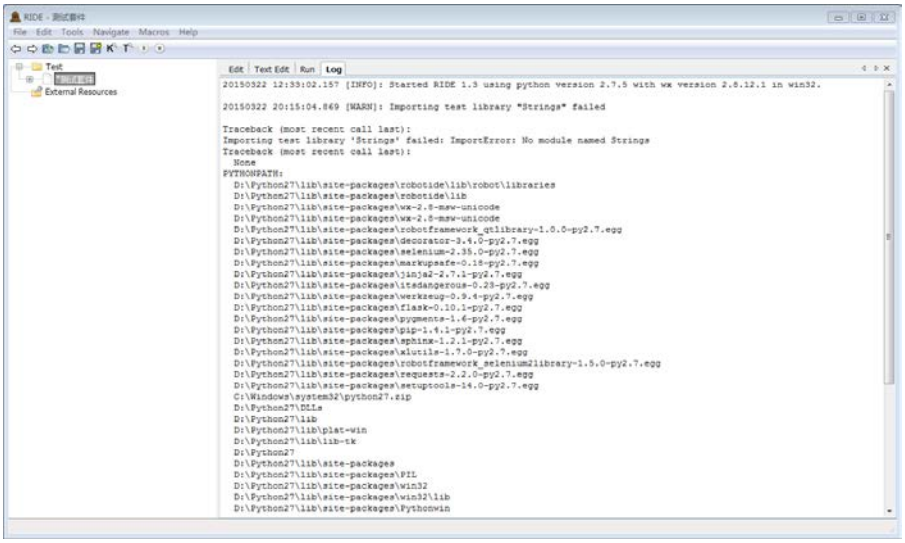

图 2-6-4

这里提示“Importing test library‘Strings’failed”，因为名字错了，所以它根本找不到这个测试库，重新把名字改为正确的 String，如图 2-6-5 所示。

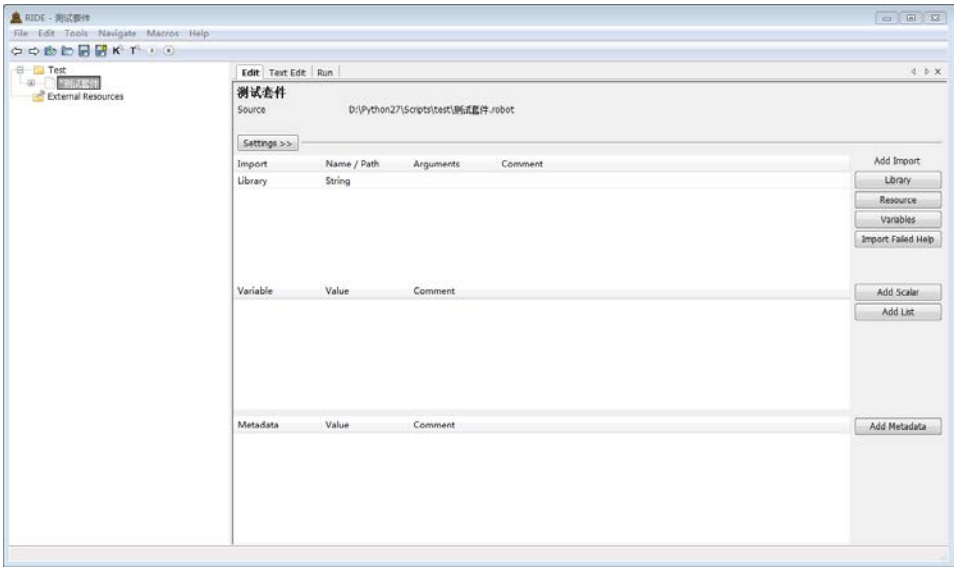

图 2-6-5

这是看到的 Library 是黑色的文字，说明加载正确。此时可以用 F5 快捷键查看一下这个测试库的关键字，如图 2-6-6 所示。

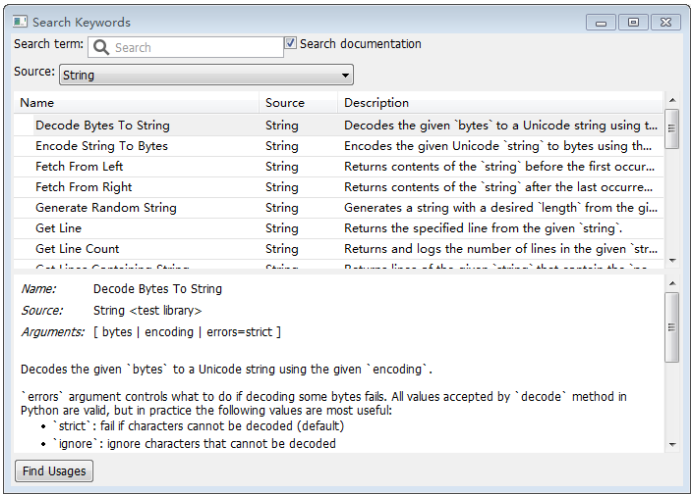

图 2-6-6

除了用测试库名字来添加测试库，也可以添加自己写的测试库，怎么自己写测试库在后面章节会介绍。这里还是拿 Robot Framework 内置的测试库来演示，它的具体位置在“PYTHON 安装目录”\Lib\site-packges\robot\libraries 目录，笔者的电脑（Windows 操作系统）上，Python 安装在 D:\Python27 目录上，所以笔者的实际目录是在 D:\Python27\Lib\site-packages\ robot\libraries，Mac 操作系统的一般是在/Library/Python/2.7/site-packages/robot/libraries 目录。通过浏览找到测试库的目录，如图 2-6-7 所示。

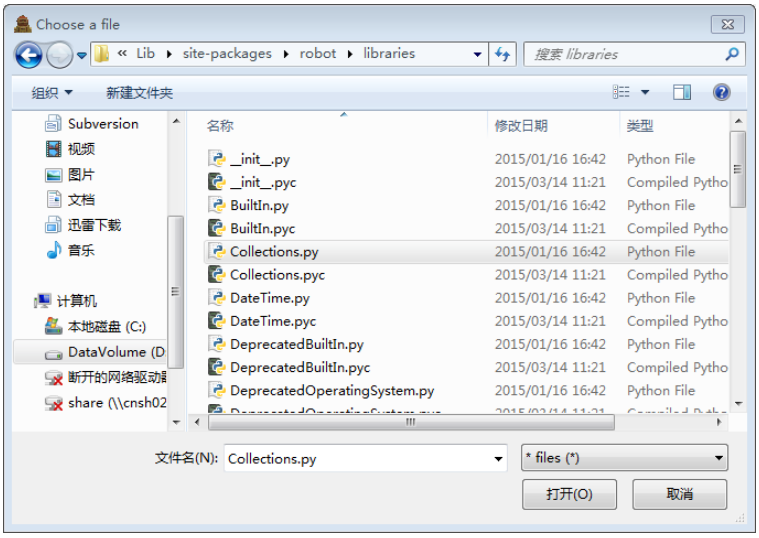

图 2-6-7

为了和 String 库有所区别，就近选了一个 Collections.py，你问为什么不选择 BuiltIn.py？因为它默认已经在 RIDE 里加载了，前面用来打印日志的 Log 关键字就是 BuiltIn 里的。

选好文件后单击“打开”按钮，如图 2-6-8 所示。

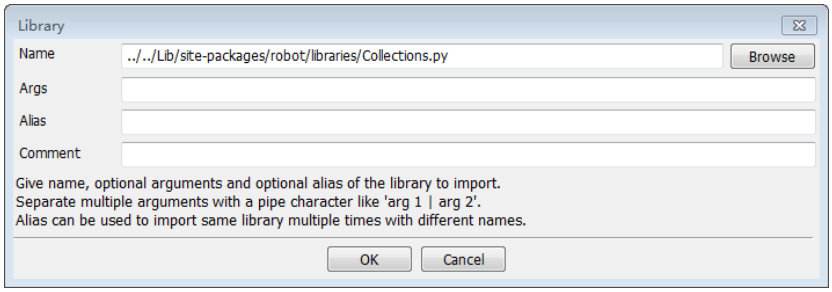

图 2-6-8

然后单击“OK”按钮，弹出的界面如图 2-6-9 所示。

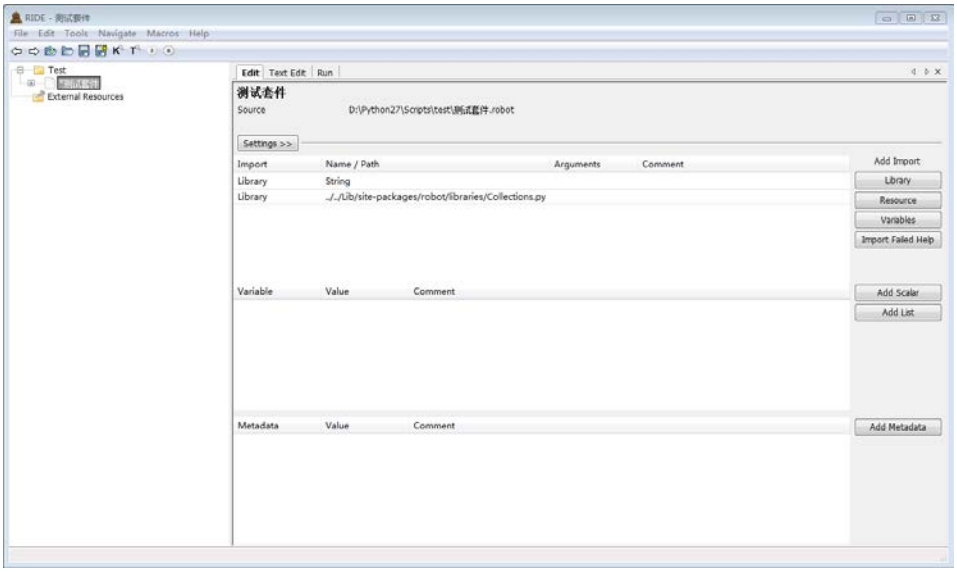

图 2-6-9

“黑色”说明加载成功了，按 F5 快捷键看一下这个测试库里有哪些关键字，如图 2-6-10 所示。

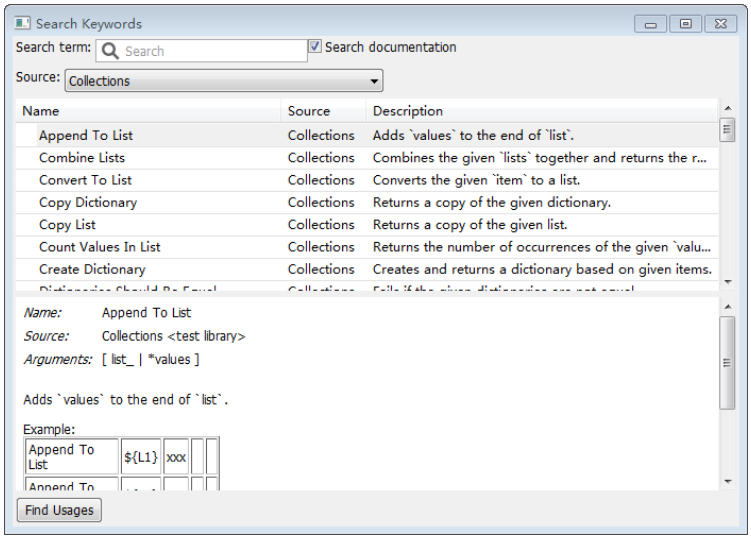

图 2-6-10

当测试库添加后，就可以使用里面的关键字了，后面会专门有一节介绍关键字的使用。

刚才添加的测试库都是基于 Python 开发的，实际上 Java 也是可以开发测试库的，不过本书不涉及 Java 这部分内容。

2.7　Resource 资源文件

2.7.1　新建资源

测试套件主要是存放测试案例，而资源文件主要就是用来存放用户关键字。

在刚才的目录型的 Project（Test Suite）下单击鼠标右键，选择“New Resource”命令，输入资源文件名称 NewResource，选择格式 ROBOT（截图默认 TXT，因为笔者的工具没有在 Tools 里改默认格式），如图 2-7-1 所示。

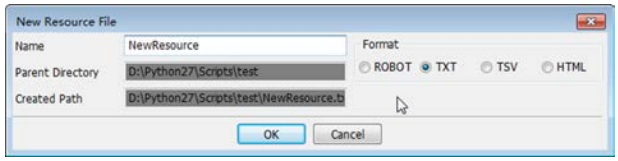

图 2-7-1

单击“OK”按钮，弹出的界面如图 2-7-2 所示。

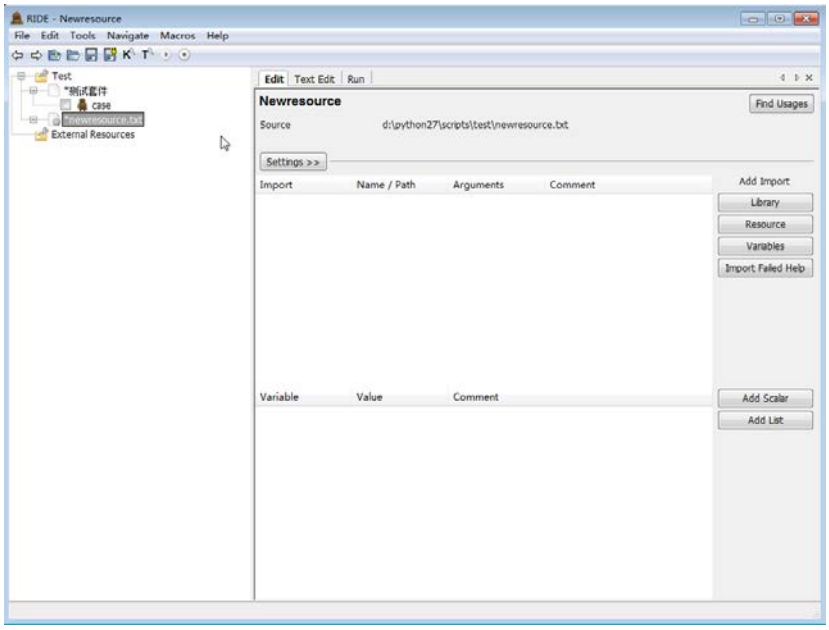

图 2-7-2

刚刚新增的时候，这个文件是灰色的，是因为它没有被任何的 Test Suite 或 Resource 加载，稍后会讲如何加载资源。

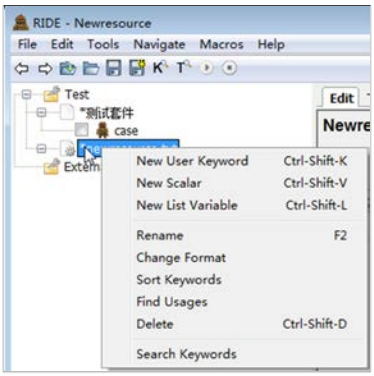

图 2-7-3

2.7.2 快捷菜单

在资源下面能做什么呢？在“资源”上单击鼠标右键，如图 2-7-3 所示。和测试套件对比，Resource 的快捷菜单多了一个 Find Usages 的命令，这个是用来查询关键字都在哪些地方使用了。资源下面能添加的主要就是 User Keyword 了，关于这个会单独讲。

2.7.3 Settings 设置项

Resources 的 Setting 界面，如图 2-7-4 所示。

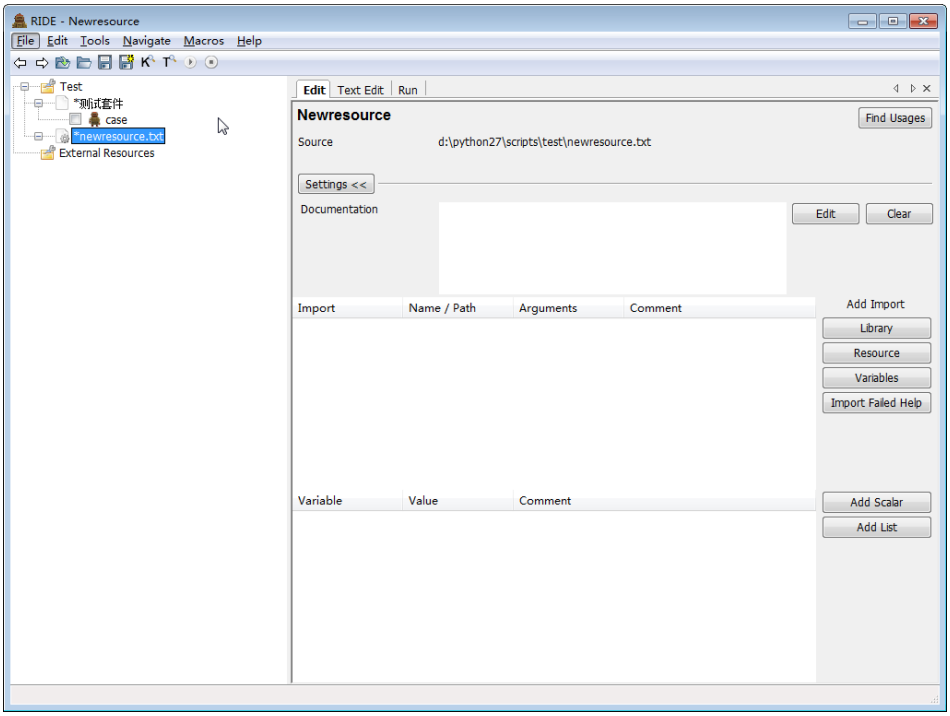

图 2-7-4

Settings 里只有一个 Documentation。Resource 右侧的工作区基本上和文件型 Test Suite 一样，多了一个“Find Usages”按钮，是查找关键字用的，另外就是少了 metadata 的部分。

2.7.4　加载资源

加载资源主要是在工作区的 Add Import 里单击“Resource”按钮来添加，因为 Test Suite 和 Resource 的工作区里都有，所以可以在这两个地方都进行加载资源。

在测试套件的工作区右侧，单击“Resource”按钮，在“Path”文本框中输入资源的完整文件名 newresource.txt（要带上扩展名.txt），如图 2-7-5 所示。

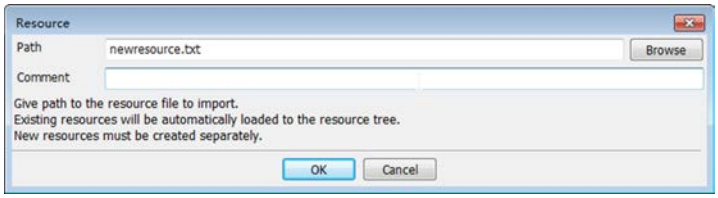

图 2-7-5

单击“OK”按钮，弹出的界面如图 2-7-6 所示。

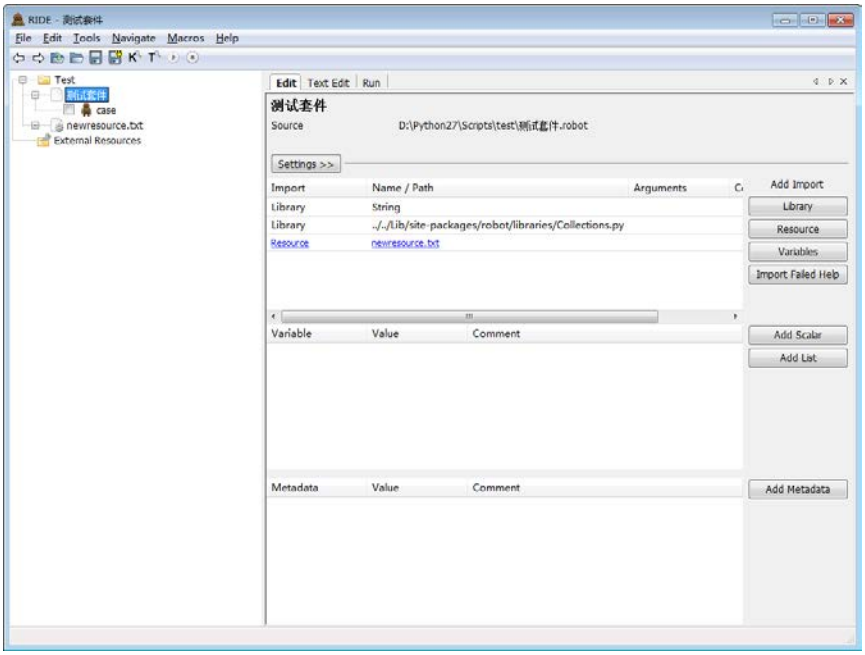

图 2-7-6

看到这个蓝色的带下划线链接的 newresource.txt 就是加载成功了。如果路径写错了或文件少了扩展名，这里会是黑色的。

2.7.5 External Resources 外部资源

外部资源的图标，如图 2-7-7 所示。主要是指不在 Project 管辖范围内的资源文件，什么样的资源文件算不在管辖范围内的呢？如果是目录的 Project，只要不在自己目录内的资源文件都算外部资源；如果是文件的 Project，它自己无法创建资源文件，其他的资源文件都算他的外部资源，即使和 Project 文件平级的。

在 External Resources 外部资源上单击鼠标右键，只有“Add Resource”命令，如图 2-7-8 所示。是用来加入已有的外部资源文件，Project 内部的资源文件就没必要再添加了，因为它已经显示在资源区了。

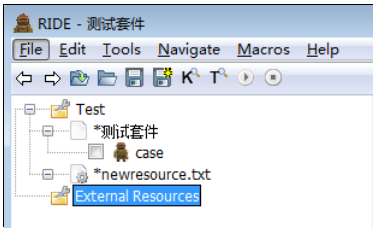

图 2-7-7

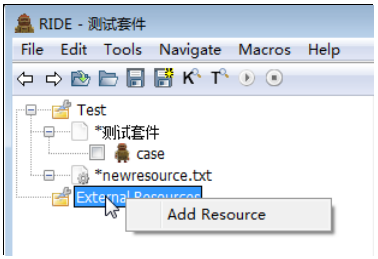

图 2-7-8

在第一个案例创建了一个 Testproject 的工程，现在又创建了个 Test 的工程，在前者我们当时没有添加资源，那么此时重新打开 Testproject 工程，在 External Resources 上单击鼠标右键，在弹出的快捷菜单中单击“Add Resource”命令，在弹出的窗口中找到 test 工程目录，打开后选择里面的 newresource.txt 资源文件，如图 2-7-9 所示。

单击“打开”按钮，这时就看到 newresource.txt 出现在了 External Resources 里，但是名字是灰色的，因为它没有被加载，如图 2-7-10 所示。

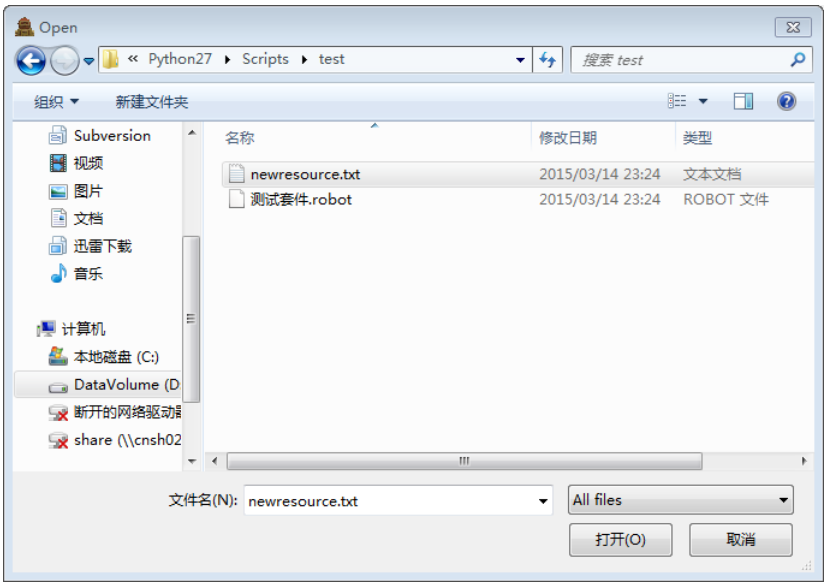
Open
Python27 ▸ Scripts ▸ test
搜索 test
组织
新建文件夹
Subversion
视频
图片
文档
迅雷下载
音乐
计算机
本地磁盘 (C:)
DataVolume (D
断开的网络驱动器
share (\\cnsh02
名称
修改日期
类型
newresource.txt
2015/03/14 23:24
文本文档
测试套件.robot
2015/03/14 23:24
ROBOT 文件
文件名(N):
newresource.txt
All files
打开(O)
取消

图 2-7-9

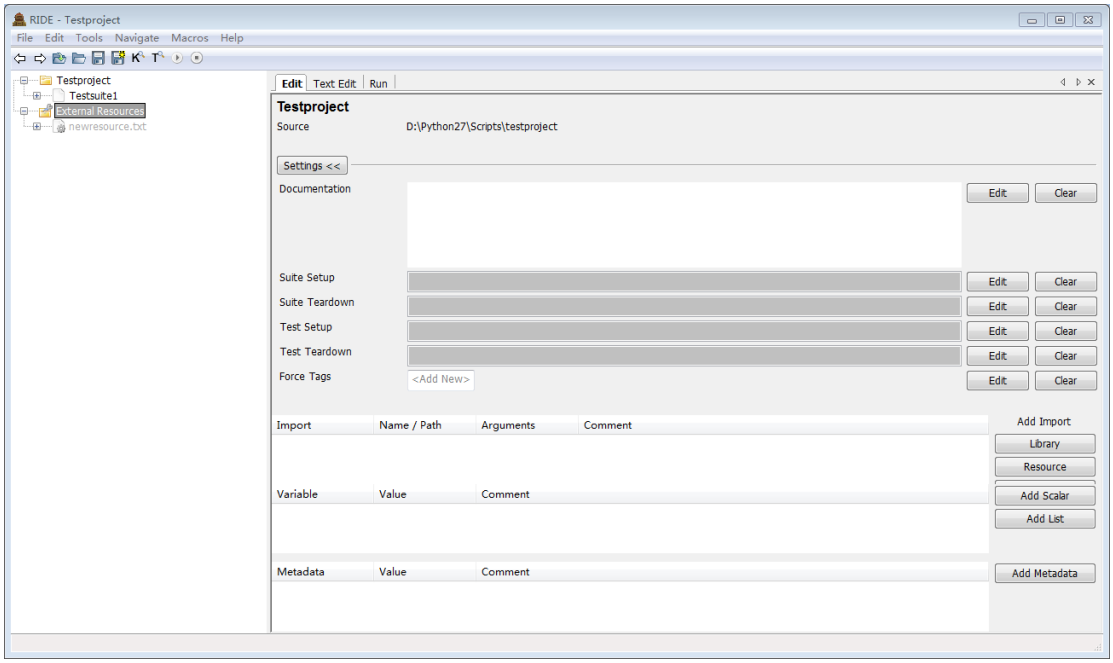
RIDE - Testproject
File Edit Tools Navigate Macros Help
Testproject
Testsuite1
External Resources
newresource.txt
Edit
Text Edit
Run
Testproject
Source
D:\Python27\Scripts\testproject
Settings <<
Documentation
Edit
Clear
Suite Setup
Suite Teardown
Test Setup
Test Teardown
Force Tags
<Add New>
Import
Name / Path
Arguments
Comment
Add Import
Library
Resource
Variable
Value
Comment
Add Scalar
Add List
Metadata
Value
Comment
Add Metadata

图 2-7-10

正确的加载方式是选中 Testsuite1，在右侧工作区单击“Resource”按钮来添加资源，这里的 Path 可以写绝对路径，也可以写相对路径，请根据实际情况进行选择。比如一个项目里的跨目录引用资源，建议使用相对路径。如果是多个项目都引用一个公共资源，可以使用绝对路径（相对路径也可以），如图 2-7-11 所示。

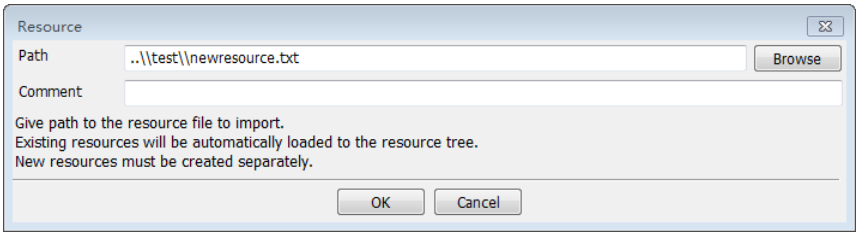

图 2-7-11

Path 里的路径分隔符“\”需要写成“\\”，因为一个“\”会被认为是转义字符的符号。当然，如果嫌麻烦也可以用“/”来分隔，比如上面的可以写成“../test/newresource.txt”，自己练习的时候可以试一下。

单击“OK”按钮添加后，如图 2-7-12 所示。

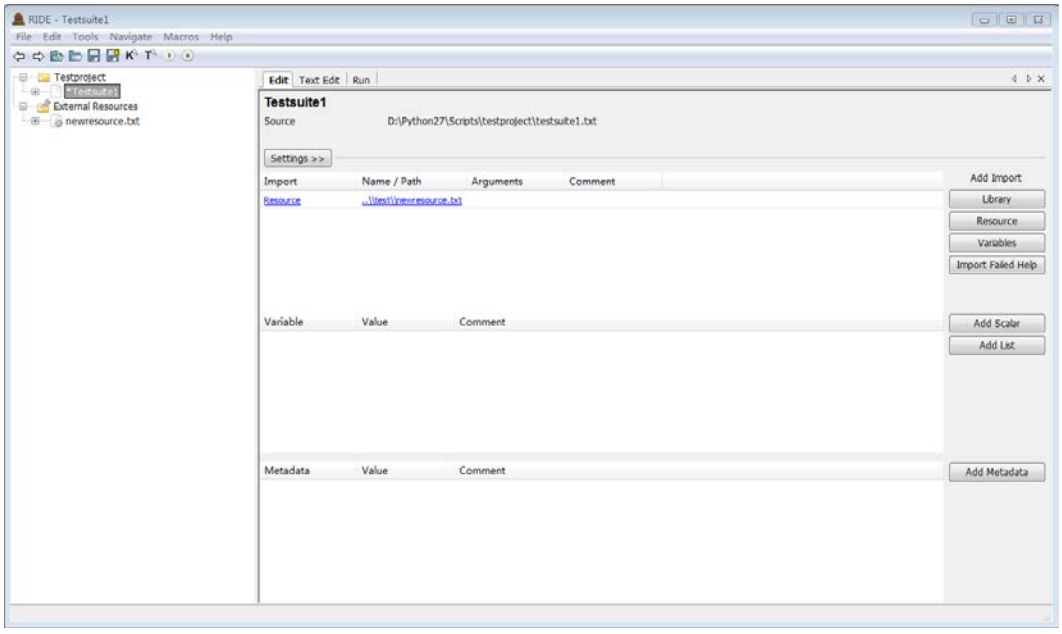

图 2-7-12

资源这里是带蓝色下划线链接的，说明加载成功了。同时 External Resources 下面的文件也变亮了，不再是灰色了。

所以要添加外部资源文件，在 External Resources 上添加只能看看，真正的引用还是要到 Test Suite 或者 Resource 那里去添加才行。

至此，再来看一下完整的工程结构关系图，如图 2-7-13 所示。

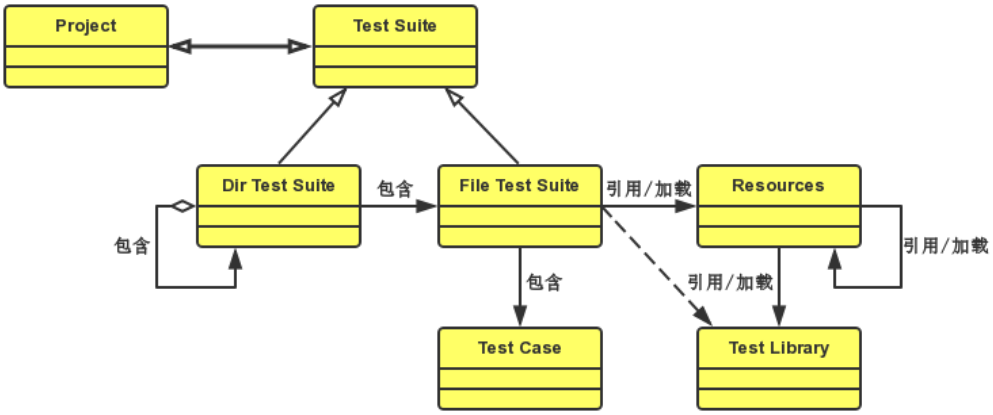

图 2-7-13

在原有的基础上增加了右边的 Resources 和 Test Library，其实在 Test Suite 去引用或加载 Test Library 也可以，但是并不推荐，所以这里用虚线标识。Test Suite 可以引用或加载 Resources，Resources 也可以加载 Resources 和 Test Library。还有一个 Variable 变量文件的加载没有画进去，它的加载可以参照 Test Library。以上就是完整的一个工程结构关系，希望能帮助大家理解自己的案例结构。

2.8　变量和常量

前面建立了 2 个工程，从这一章节开始，先用 Test 那个工程继续给大家讲解，每一个章节会创建一个 Suite，每一个小节会创建一个 Case，方便大家渐进式地学习，同时工程的源码，笔者会放到 github 上分享给大家。

这一章节的例子是 Suite-2-8，Case 名字与小节相同，如 Case-2-8-1。至于创建 Suite 和 Case 的过程就略过了，可以参考前面的章节。在 Case 里的脚本，除了用图来展示，笔者也会加上表格来展示脚本。

2.8.1 变量与常量基础

1. 变量标识符

每个变量都可以用变量标识符{变量名}来进行表示，变量主要有两类，一类是 Scalar；另一类是 List。Scalar 型变量用“$”作为标识符；List 型变量用“@”作为标识符。例如：${var}, @{lvar}，就分别是 Scalar 变量和 List 变量的展现形式。

从标识上可以先初步这样区分，而对于这两种变量来说，List 大家都很容易理解，那么什么是 Scalar 呢？从翻译上它的中文名叫标量，按笔者个人的理解，更愿意称之为单值变量，与之对应的就是 List 这种多值变量了。单值并不仅仅是像字符串、数值这样的，还有可能是个对象或者是个字典。只有带@标识的变量能够确认是 List 型变量，而带$标识的变量则要取决于它的变量名或者变量值，才能确定它到底是什么变量，特别是变量值。因为变量之间是可以转换的，例如使用“$”标识的变量，实际上也可以在接收 List 值后转换成 List 变量。转化的内容会在转换的章节里介绍。

2. 变量声明

其实在 RF 里没有什么特别的变量声明，因为 RF 的底层是 Python，所以它的语法也有些类似，变量不需要特定声明，只要有初始化赋值即可使用。

如果硬要说有声明，那可以把在 TestSuite 下面手动添加的变量理解为声明。比如可以在 TestSuite 上单击鼠标右键，或者在 Edit 区单击“Add Scalar”或“Add List”按钮来新增变量，这里分别添加一个看一下效果。

单击“Add Scalar”按钮，如图 2-8-1 所示。

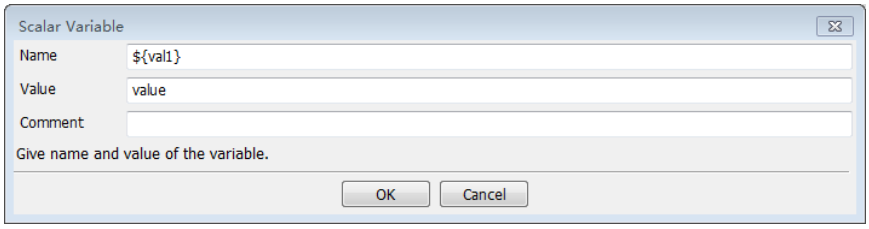

图 2-8-1

单击“Add List”按钮，如图 2-8-2 所示。

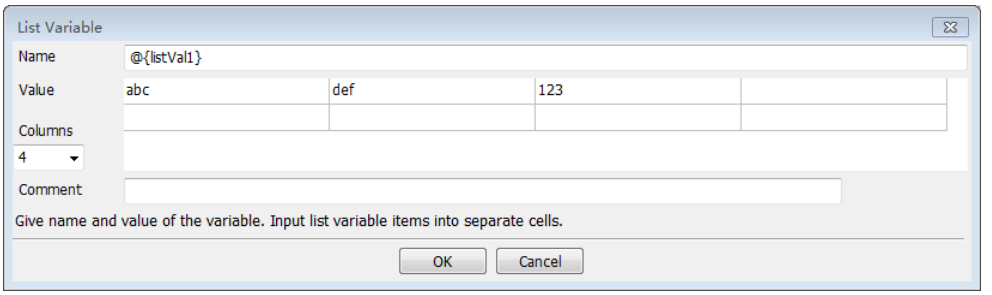

图 2-8-2

最终都添加完成后，如图 2-8-3 所示。

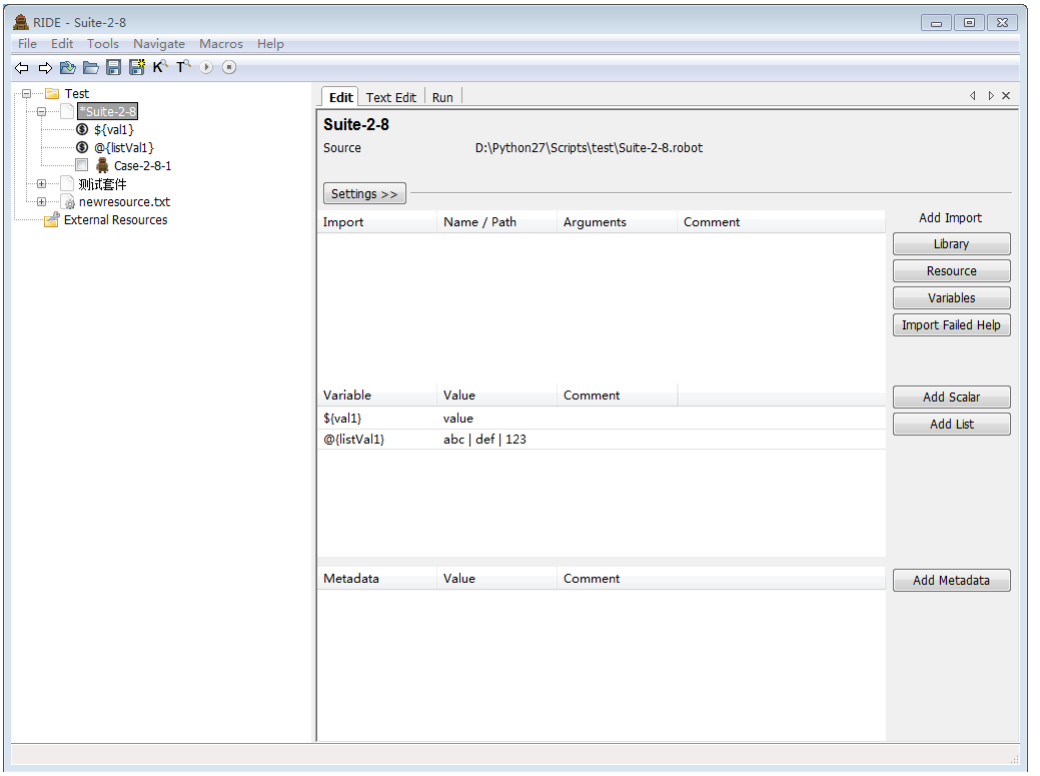

图 2-8-3

实际上，这样手动添加也可以看作另一种形式的变量赋值，和后面讲的变量赋值用 Set Variable 是一样的。无论是 Scalar 变量还是 List 变量，都可以用 Set Variable 来赋值，这个例子放到具体的变量中再讲解。

在 Case-2-8-1 里把这两个变量的值打印一下，写完的脚本见表 2-8-1。

表 2-8-1

log	${val1}	
log many	@{listVal1}	

这里用了两个关键字 log 和 log many，分别适用于 Scalar 变量和 List 变量，如果用错了，系统会报错的。可以在自己练习时试试把两个关键字后面的变量互换一下，这里就不展示了，原因在接下来的章节里会介绍。

勾选这个案例运行一下，运行结果如图 2-8-4 所示。

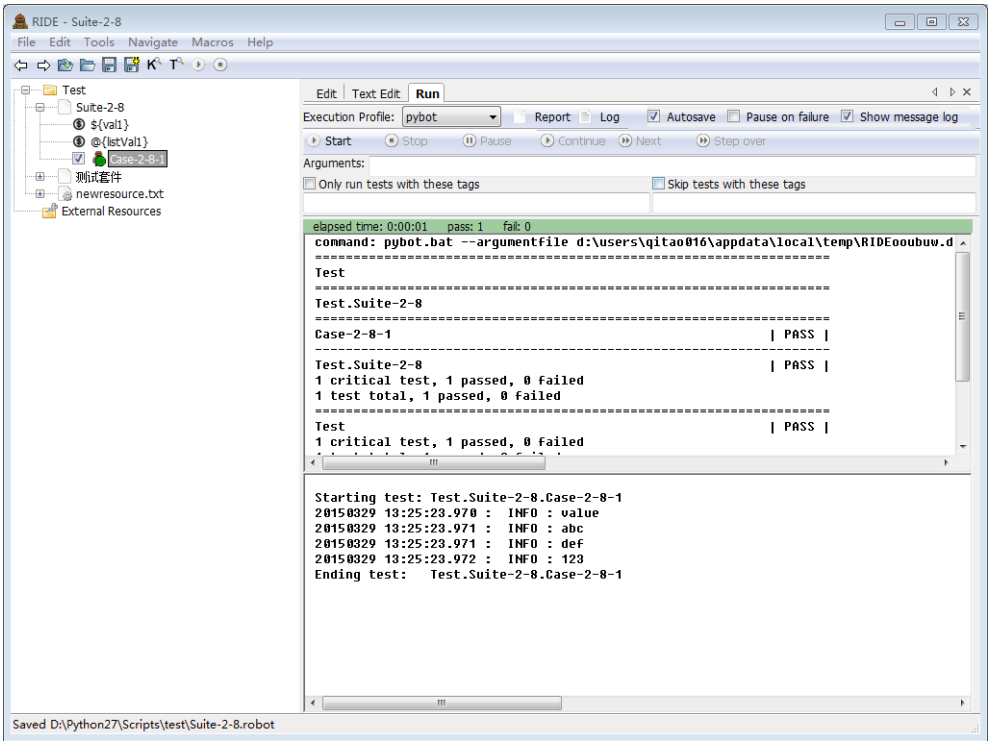

图 2-8-4

可以看到这两个变量的值都打印出来了，只不过@{listVal1}的 3 个值是分别打出来的。

3. 变量的作用域

在通常情况下，每个变量默认都是局部变量，在前面各个组件的快捷菜单里都有看到

添加变量的操作，那么在默认情况下，这些变量的作用域为：

- 一个 case 里的变量，作用域在这个 case 内部；
- 一个 userkeyword 里的变量，作用域在这个 userkeyword 内部；
- 一个文件型 suite 里的变量，作用域在这个 suite 内部，所有下面的 case 也都可以使用；
- 一个目录型 suite 里的变量，作用域在这个目录内，但也只是目录内，它下面的文件型 suite 是无法使用的，所以一般在目录下新增变量没有太大意义。

变量的作用域是可以改变的，通过一些关键字的处理，对变量进行作用域的改变，常用的关键字有：

- Set Global Variable 为设定全局级变量。当执行过这个设置后，这个变量在所有的测试案例和测试套件里都有效；
- Set Suite Variable 为设定 File Suite 级变量。当执行过这个设置后，这个变量只要在当前的 File Suite 内都有效，包括下面的案例；
- Set Test Variable 为设定 Case 级变量。当执行过这个设置后，这个变量在当前 Test Case 内有效。

由于验证这个作用域还涉及一些没有讲到的知识，所以笔者决定这里只是把说明列在这里，在第二章的最后一个章节里再来做这个验证。

最后，建议大家尽量少用这几个改变变量作用域的关键字，特别是在你还没有理解的情况下，会把你绕晕的。

4. 常量

除了变量以外，RF 里还有一些常量。常量主要有环境变量、数值常量、特殊字符常量、系统保留变量。其中环境变量的标识符为“%”，其他的几个标识符与 Scalar 变量相同，都是“$”，所以笔者前面才说，单值变量要取决于它的变量名和变量值，才能知道它到底是什么变量。

（1）环境变量：无论是在 Windows 操作系统中，还是 Mac 操作系统中，都可以定义自己的环境变量，定义方式和在自己的命令行里使用的方式可能都不同。但是在 RF 里，都可以统一用标识符“%”来使用，例如笔者的这个 Windows 操作系统中定义了一个环境变量 ANDROID_HOME，如图 2-8-5 所示。

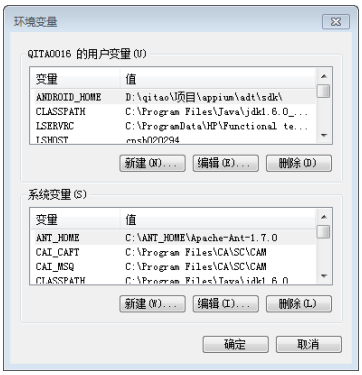

图 2-8-5

那么可以在 Case 里加一行脚本，打印一下这个环境变量%{ANDROID_HOME}，脚本比较简单，见表 2-8-2。

表 2-8-2

log	%{ANDROID_HOME}

运行一下这个案例，运行结果如图 2-8-6 所示。

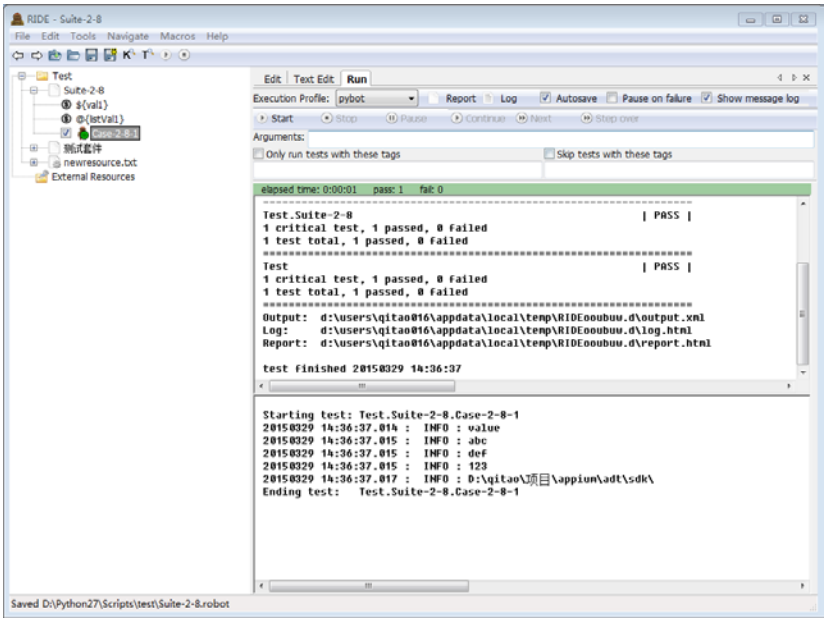

图 2-8-6

作为环境变量，你只能去定义它的地方修改它的值，在 RF 里只是使用，不能赋值，虽然叫变量，实际上属于常量了。

（2）数值常量：在通常情况下，在 RIDE 里的所有字符都会被当作字符串，即使变量的值是数值，默认也是字符串形式存在。如果想要让它直接以数值的方式存在，就可以用到数值常量。例如数值 2.6，可以写成${2.6}。

这里用一个例子来展示数值常量和默认字符串的区别，这个例子如果看不懂不要紧，因为笔者提前引入了一点后面章节的内容。例子见表 2-8-3。

表 2-8-3

${shuzhi}	Set Variable	${2.6}	2.6

运行一下就能看出区别了，运行结果如图 2-8-7 所示。

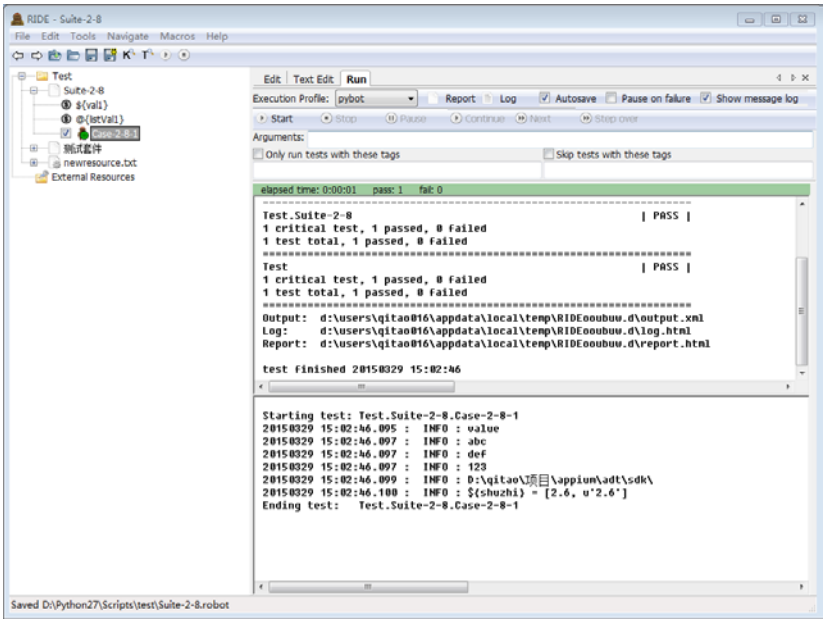

图 2-8-7

这里其实笔者是定义了一个 List 变量，用 Scalar 变量的形式展示出来，大家可以看到它有 2 个元素，用${2.6}赋值的就是以数值常量来赋值了，而直接写的 2.6，在这里被当作 unicode 字符串 u‘2.6’。其实用数值常量或者直接写的差别不大，在进行数值运算时，其实会强制转换的。

（3）特殊字符常量和系统保留常量：这两类常量你平时不知道它们在哪里的，只有一个办法可以把它们找出来，那就是 Content Assistance 内容助手，前面介绍菜单时介绍过，它的 2 个快捷键，笔者个人比较喜欢“Ctrl+空格”组合键，在 Windows 操作系统中使用要改一下，在 Mac 操作系统中一般也会冲突，可以使用另一个快捷键或者改一下。先看看都有哪些常量，在 Case 里双击一下空白的单元格，此时有光标闪烁，说明进入了编辑状态，此时按“Ctrl+空格”组合键（如果你的这个组合键和其他组合键冲突，可以按另一个“Ctrl+Alt+空格”组合键），如图 2-8-8、图 2-8-9 和图 2-8-10 所示。

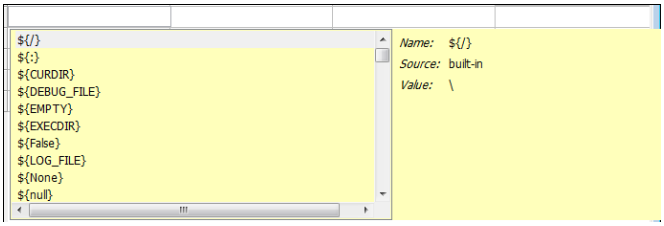

图 2-8-8

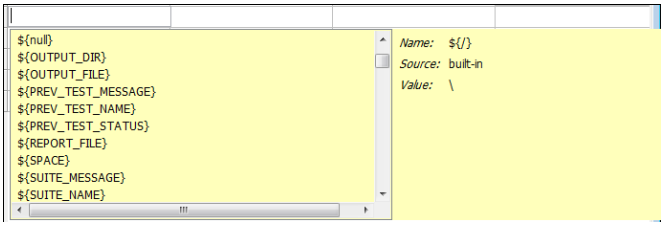

图 2-8-9

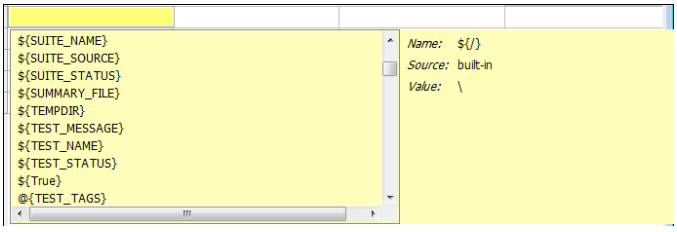

图 2-8-10

特殊字符常量的主要有${/} ${:} ${EMPTY} ${False} ${None} ${null} ${SPACE} ${True}，其他的都是系统保留常量。

这里面每个常量的值都可以在右边的提示框里看到它的 Value，可以使用键盘的上下键，切换每一个常量查看它的值，如果值为空的（除了 EMPTY）都是在运行时才有它的

值。这里就不展开细说了，以后的实例中应该会穿插使用到一些的。

2.8.2　Scalar 变量

1. 变量赋值

变量赋值也是有几种的，根据自己实际情况来使用。

（1）Set 赋值

通常最常用的方式主要是使用 Set Variable 关键字对变量进行赋值，其他 Set 相关的带 Variable 的关键字也可以进行赋值。

赋值的时候，变量后面写不写“=”都可以，例如见表 2-8-4 代码。

表 2-8-4

${val2}	Set Variable	abcd		
${valif2}	Set Variable If	'${val2}'=='abcd'	efgh	ace

用了 2 个例子，第一个变量赋值为 abcd。第二个变量判断如果${val2}的值和 abcd 字符串相等，就赋值 efgh 给${valif2}；如果不相等则赋值 ace。如图 2-8-11 所示，图中第 5 列看不到，以表格里的脚本为准。

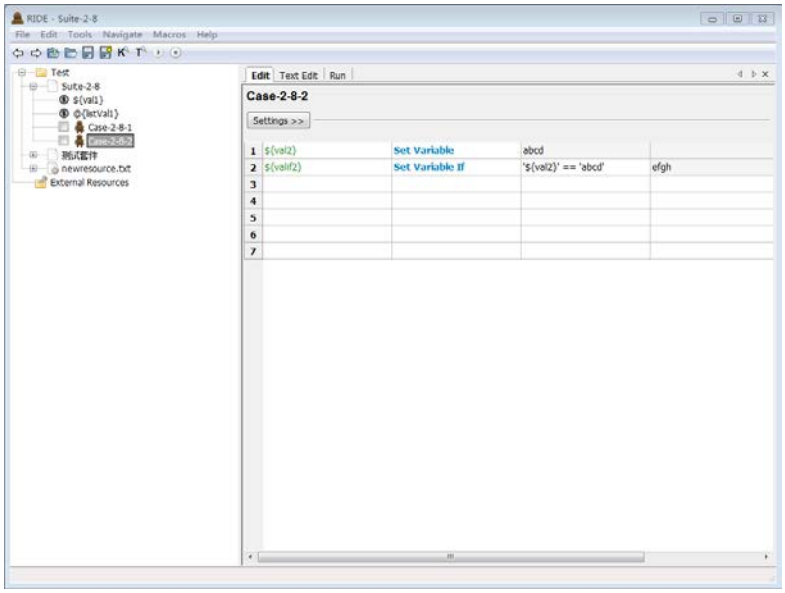

图 2-8-11

运行结果如图 2-8-12 所示。

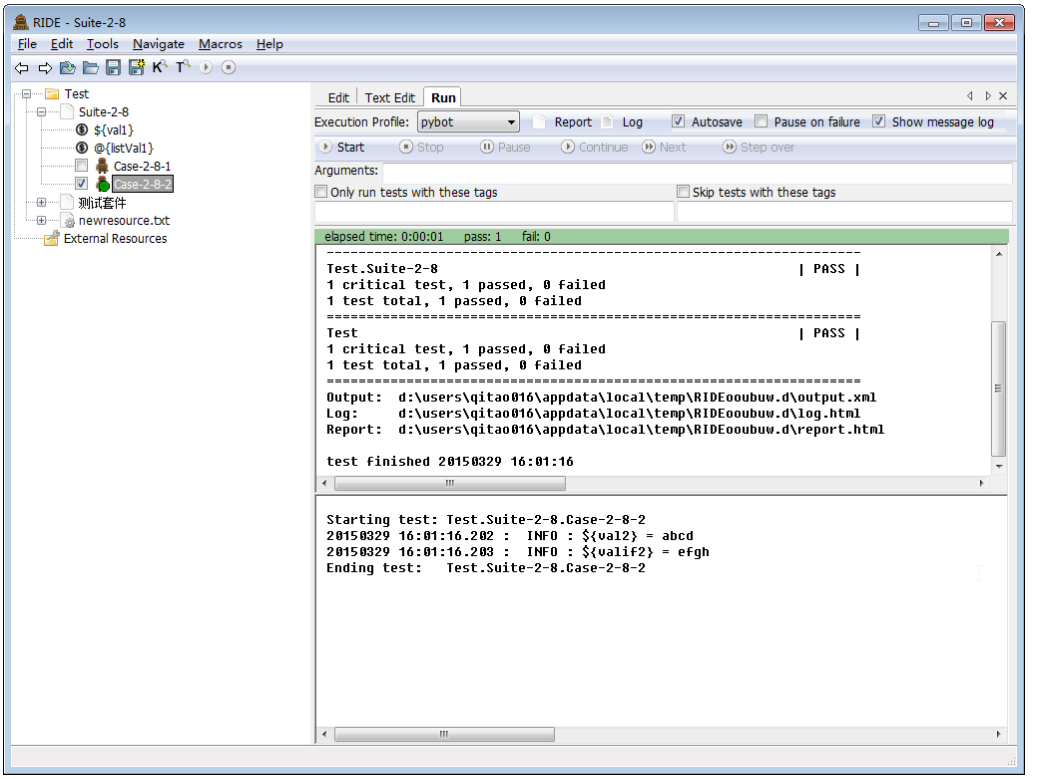

图 2-8-12

（2）Get 赋值

主要用于返回值上，包括系统关键字的返回值和用户关键字的返回值（从某种意义上讲，Set 那个也是返回值），案例见表 2-8-5。

表 2-8-5

${getVal1}	Get Length	${val2}
${getVal2}	Get Time	

接着前面的例子，第一个例子是获取变量${val2}的长度，第二个例子是获取时间。运行结果如图 2-8-13 所示。

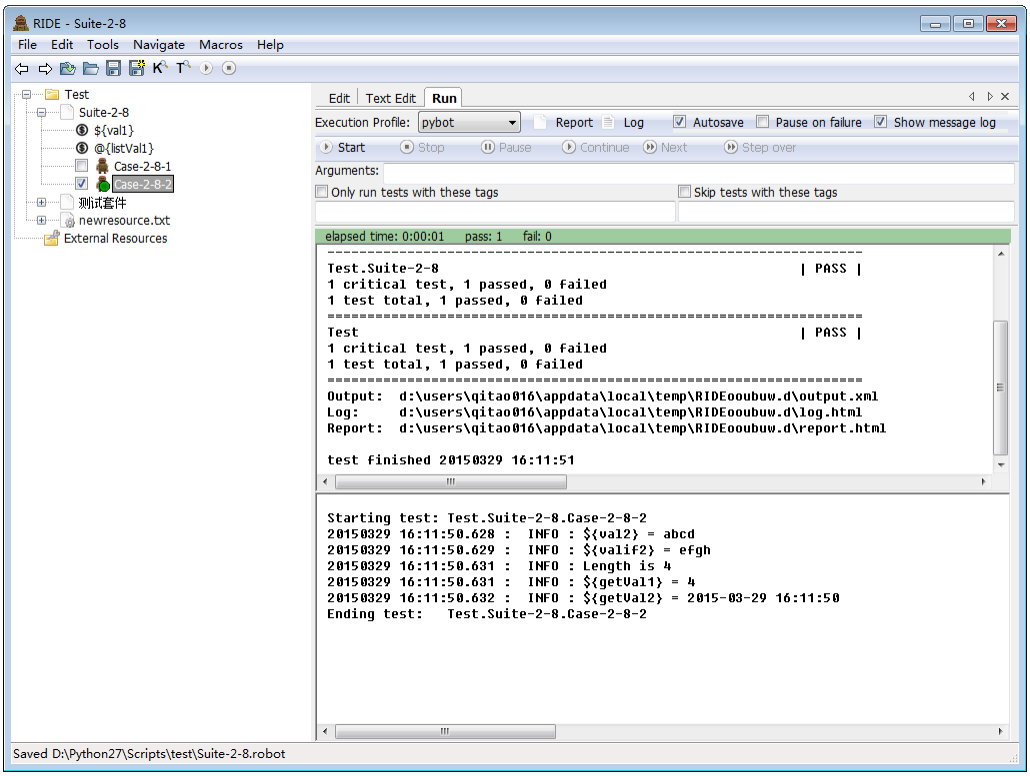

图 2-8-13

后面讲到用户关键字里也会有一些例子，用法是一样的。

（3）命令行赋值

在运行时使用“-v”的参数来给变量赋值，比如在案例中加上打印${val1}变量日志的脚本，见表 2-8-6。

表 2-8-6

log	${val1}

如果直接运行，就会是最初定义的变量值 value 了（在变量声明那里），此时在 Run 的界面上的 Arguments 里加上一行:

```
-v val:running
```

如图 2-8-14 所示，因为是连着前面做的，所以界面上还有前一次的执行结果，请只关

注 Arguments 那里。

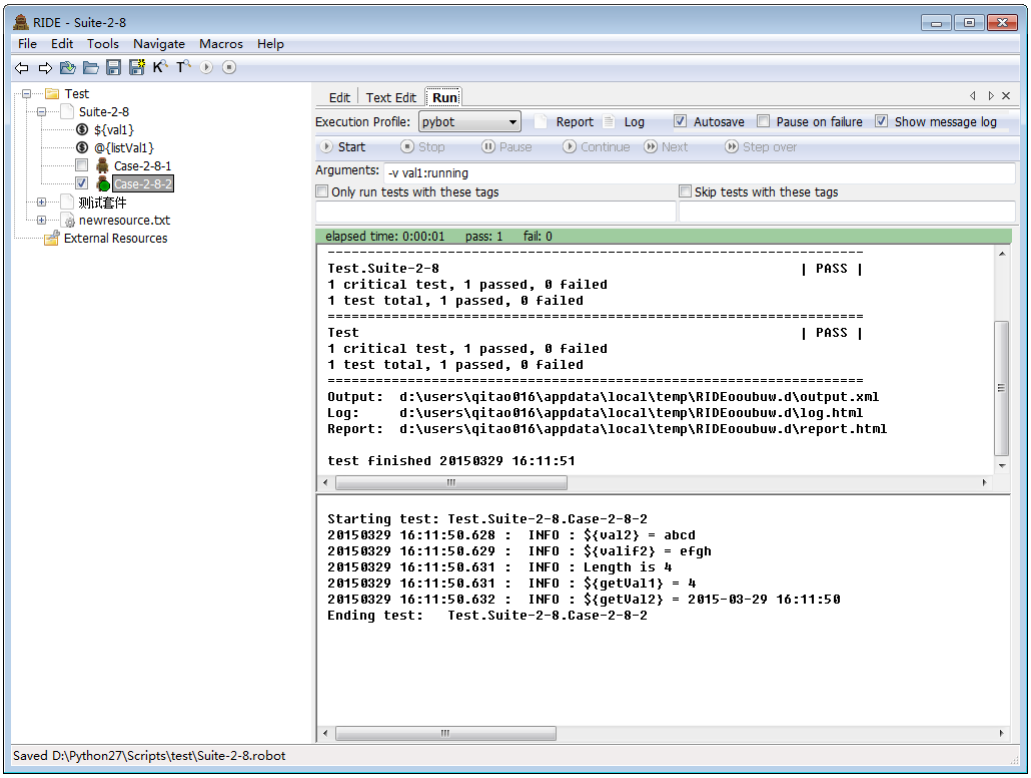

图 2-8-14

此时再运行案例，运行结果如图 2-8-15 所示。

可以看到${val1}的值已经被改成了 running 了。

tips：如果一个变量没有经过初始化或赋值，使用时会报错的。

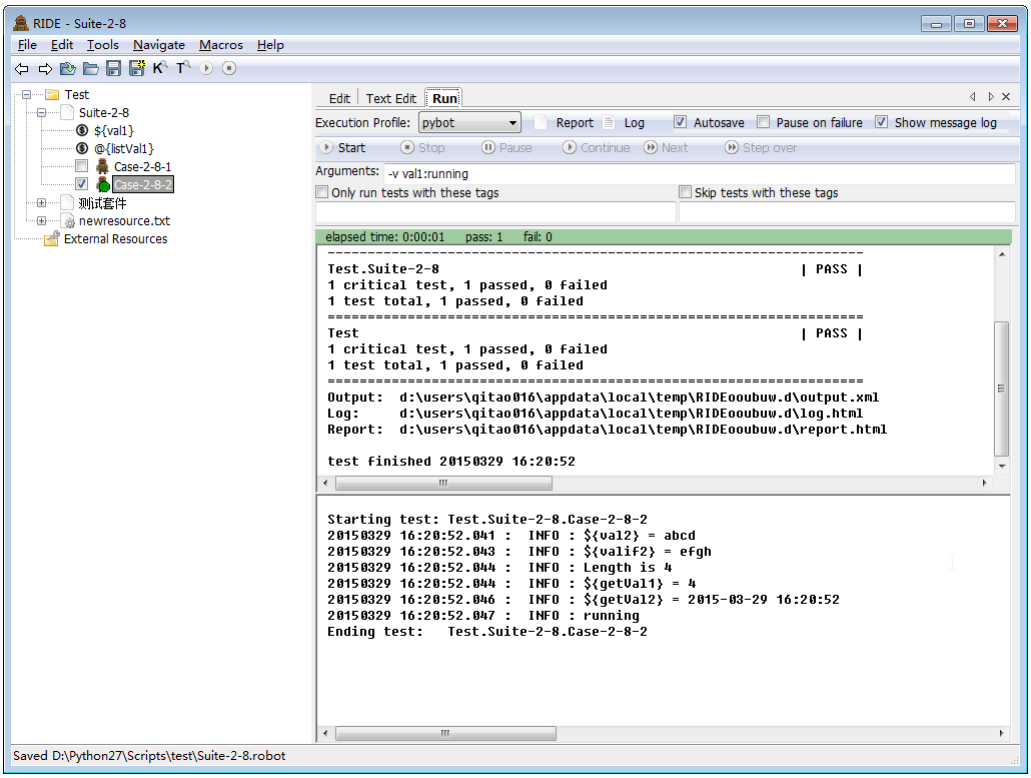

图 2-8-15

2. 变量使用

其实前面已经有过一些使用了，这里列几种常见的使用。

（1）在判断中使用

刚才在变量赋值里，其实已经有一个 Set Variable If 的例子了，这里换一个例子，见表 2-8-7。

表 2-8-7

Run Keyword If	'${val2}'=='abcd'	log	efgh

这个关键字的意思是说，如果${val2}和 abcd 字符串相等，那就运行后面的 log 关键字，打印 efgh。运行案例，运行结果如图 2-8-16 所示。

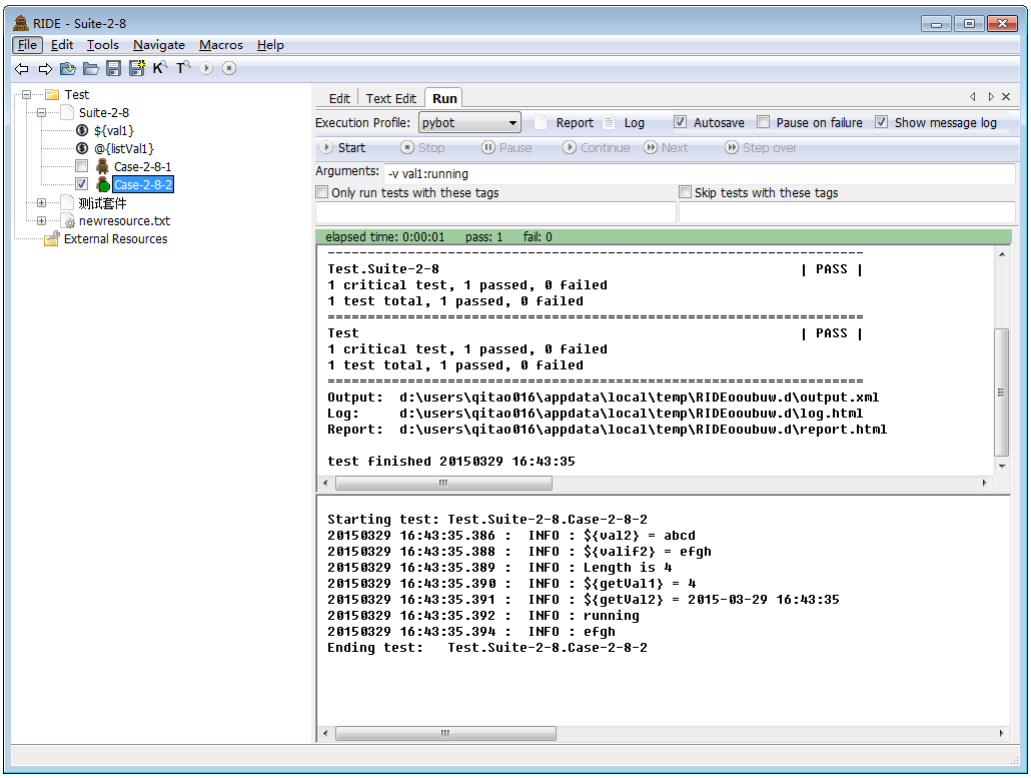

图 2-8-16

因为现在的条件成立，于是运行结果如预期的打印了 efgh，假设条件不成立，则不会做任何操作。

（2）字符串使用

如果想把变量作为一个字符串的一部分，可以直接这样写，见表 2-8-8。

表 2-8-8

log	0123${val2}efgh	

运行结果如图 2-8-17 所示。

实际上在默认情况下，RF 里的变量都是字符型的，并且两个字符串连接不要加什么符号，直接连起来就行了。如果你写成 0123+${val2}efgh，最终结果就是 0123+abcdefgh，它会把你写的任何内容都当作字符串的。

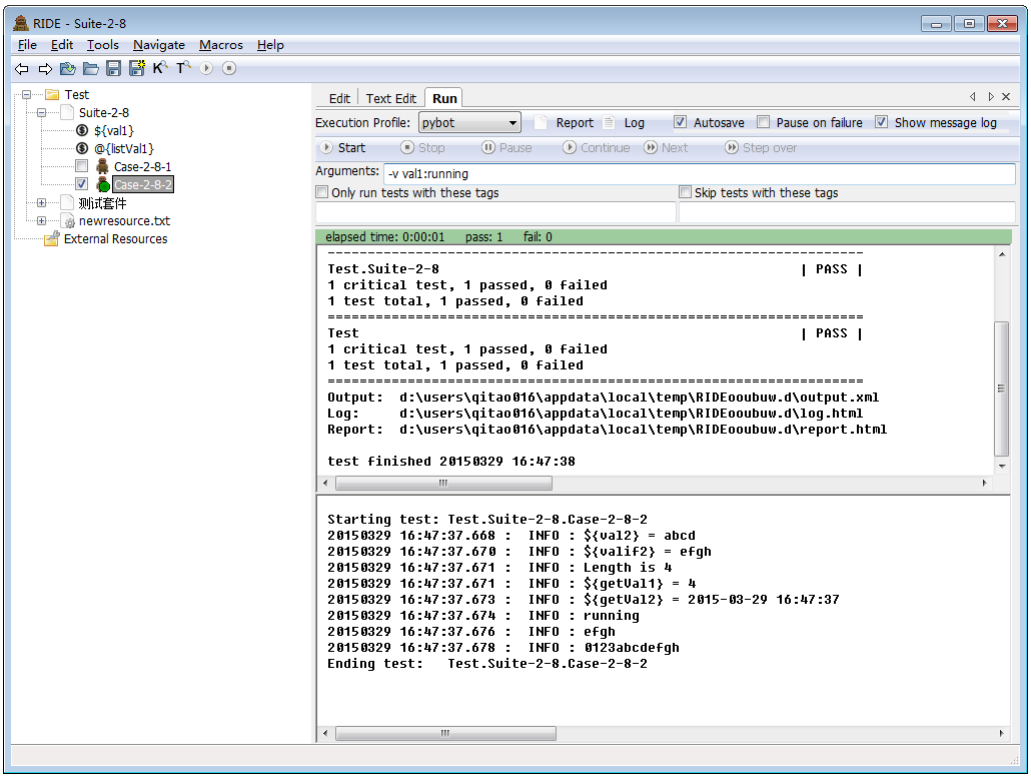

图 2-8-17

除了字符串连接，还有字符串截取。比如还是这个${val2}，如果想截取它的第 3 个字符，还有第 1 个到第 3 个字符，脚本可以写成这样，见表 2-8-9。

表 2-8-9

log	${val2[2]}	
log	${val2[0:3]}	

可以看到“[]”里写的是字符串的 index 索引位置，abcd 第 3 个字符 c 的 index 是 2，因为 Python 里的 index 是从 0 开始的，所以要写成“[2]”；而截取第 1 个到第 3 个字符写的是“[0:3]”，其实它相当于“[0:2]”或“[0:3)”，也就是 0 在区间内，3 不在区间内。运行结果如图 2-8-18 所示。

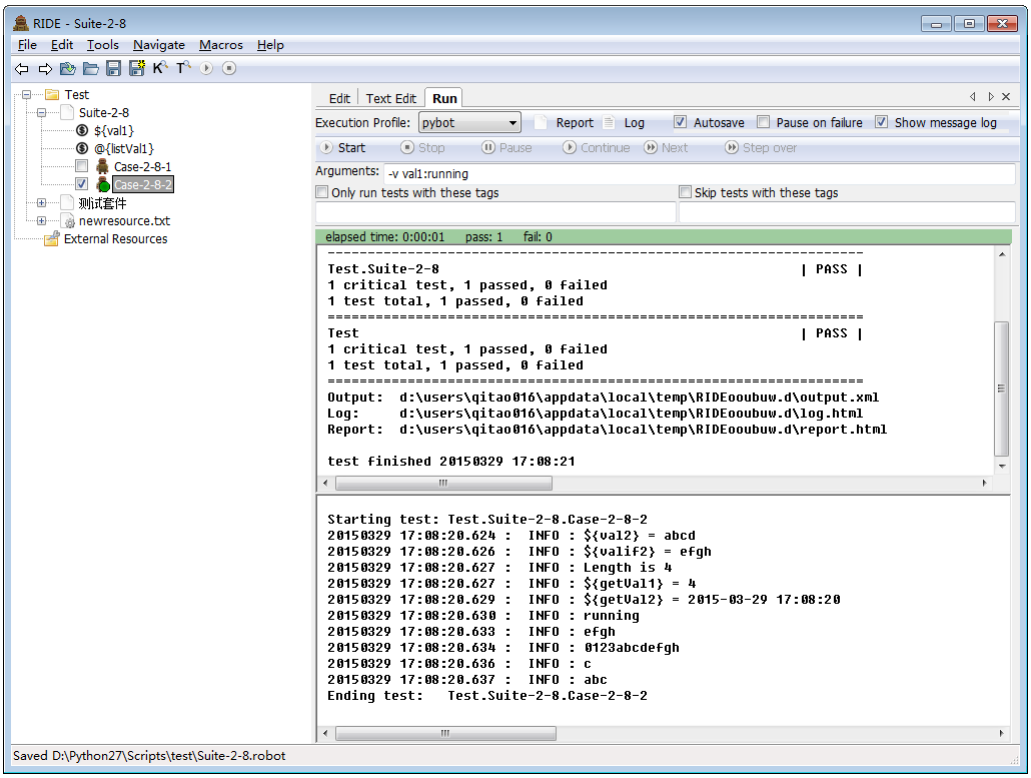

图 2-8-18

所以看到的“[2]”的结果是 c，“[0:3]”的结果是 abc。

（3）参与运算

前面说的都是字符的，现在看看数值型的变量该怎么运算。

这里要用到一个关键字 Evaluate。先看个例子，见表 2-8-10。

表 2-8-10

${cal1}	Set Variable	123
${cal2}	Evaluate	${cal1}+1

这里给变量赋值为 123，然后用 Evaluate 让它进行加 1 的运算，运行结果如图 2-8-19 所示。

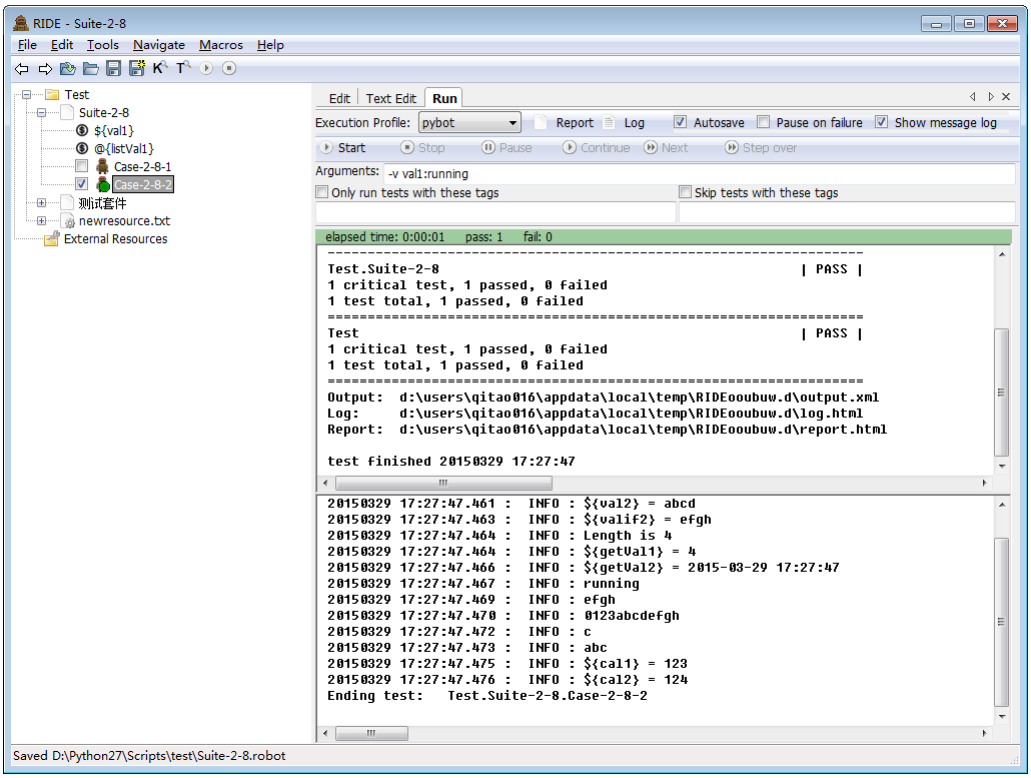

图 2-8-19

但是有时候可能你的变量值是字符串形式的数值，见表 2-8-11。

表 2-8-11

${cal1}	Set Variable	'123'
${cal2}	Evaluate	int(${cal1})+1

这里给变量赋值为字符串'123'，然后用 Evaluate 让它进行加 1 的运算，但是不能像刚才一样直接加 1 了。要先强制转换成 int 类型才能计算，运行结果如图 2-8-20 所示。

Evaluate 这个关键字用处非常大，它实际上是将后面的表达式放到 Python 里进行运行，在笔者的博客上将它称为“万能的 Evaluate”，后面专门有一个章节留给“万能的 Evaluate”。

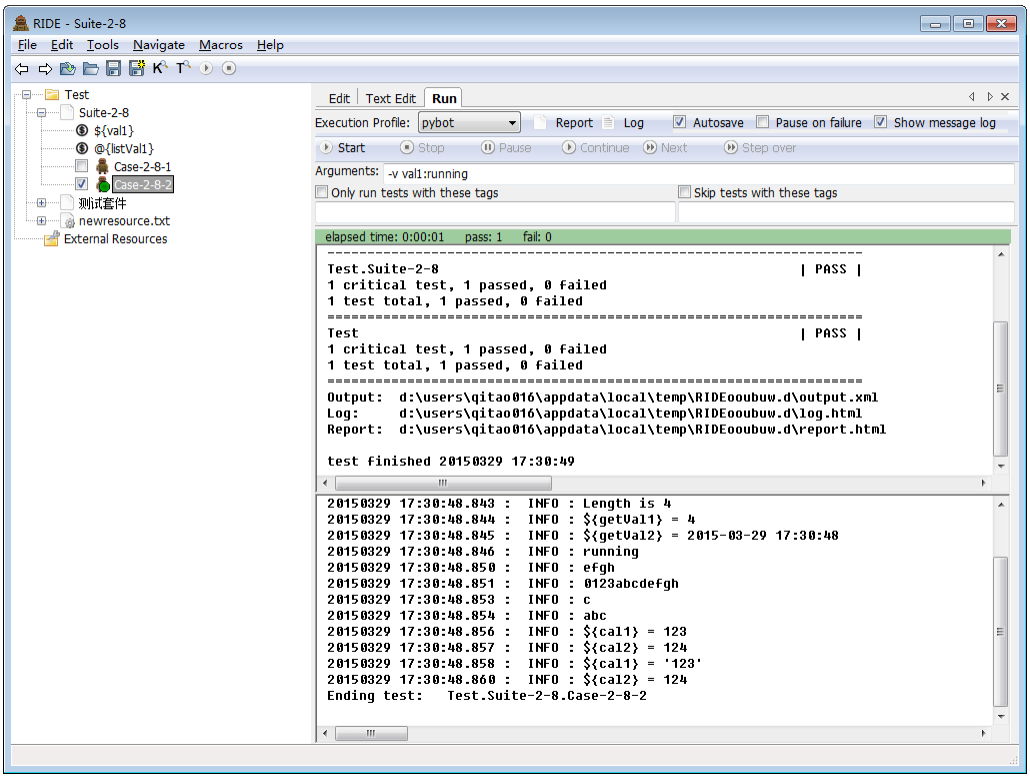

图 2-8-20

2.8.3 List 变量

1. 变量赋值

和 Scalar 类似，List 变量也可以用 Set Variable 来赋值，但是它最正式的赋值还是用 Create List。可以看一下 Set Variable 的关键字说明，如图 2-8-21 所示。

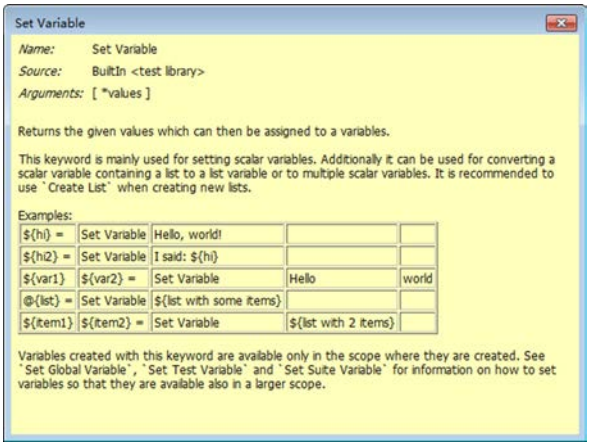

图 2-8-21

这里提到 Set Variable 这个关键字，主要用于设置 Scalar 变量。同时它也能用来

把一个 List 变量或者多个 Scalar 变量形成的 List 转换给一个 Scalar 变量，通常推荐使用 Create List 来创建新 List。说明里也给出了一些例子。

这里做个简单的例子，脚本见表 2-8-12。

表 2-8-12

@{Val3}	Set Variable	1	2	3
@{listVal3}	Create List	3	2	1

这里分别用 Set Variable 和 Create List 创建 2 个 List 变量，运行结果如图 2-8-22 所示。

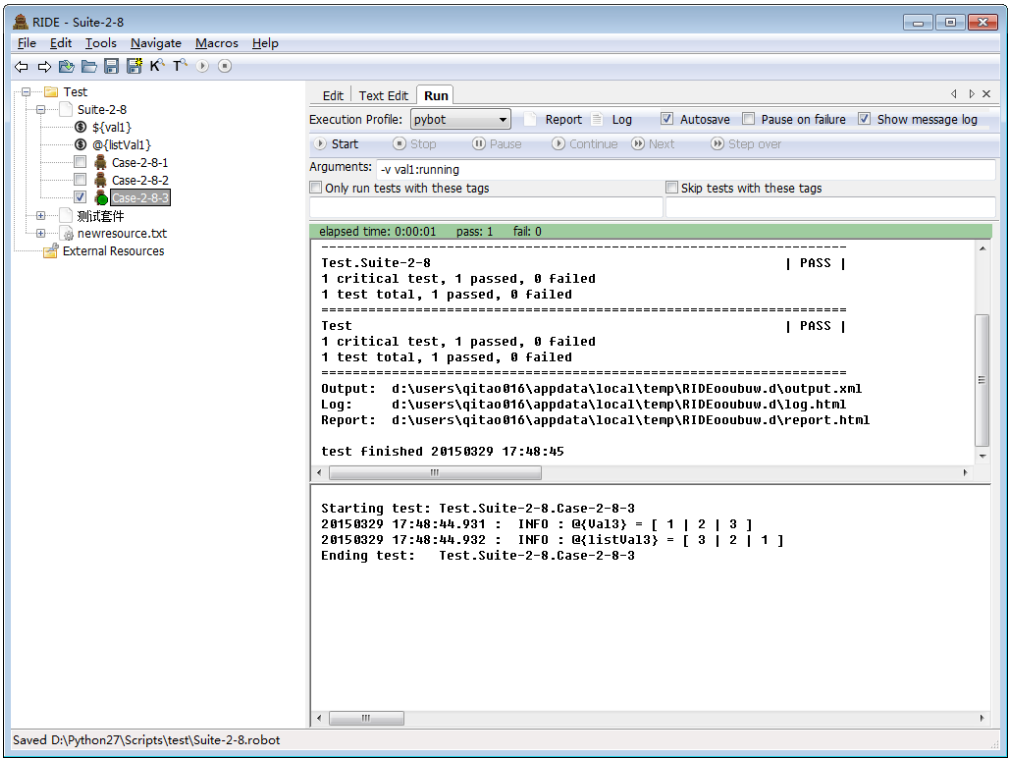

图 2-8-22

2. 变量使用

在使用上，最直观的是看关键字的参数到底是 Scalar 的还是 List 的，区别就是看变量名前面是否有“*”（星号），刚才使用过一个关键字叫 Run Keyword If，这次用另一个关键字 Run Keyword 来进行演示。先看一下这个关键字的说明，如图 2-8-23 所示。

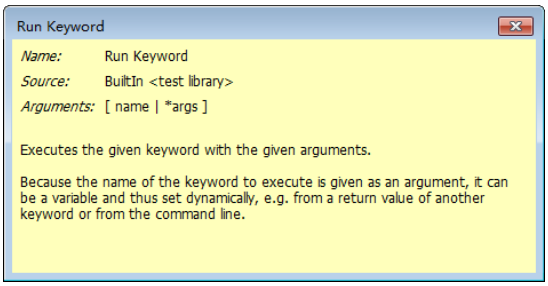

图 2-8-23

这个关键字有两个参数，name、*args。name 就是支持传入 Scalar 变量的参数；*args 就是支持可以传入 List 变量的。这个关键字的作用就是传一个 name 的参数，这是要执行关键字的名字，*args 就是要执行关键字的参数，因为每个关键字所需要的参数个数是不定的，所以这里用了可变个数的参数。

List 变量其实是不定个数的，它有几个元素，就相当于有几个 Scalar 变量，所以对于 *args，读者可以按照关键字需要的参数数量一个一个地写，也可以传一个 List 变量。

接下来看另一个关键字 Log，用了很久也没有介绍过它，先看看关键字说明，如图 2-8-24 所示。

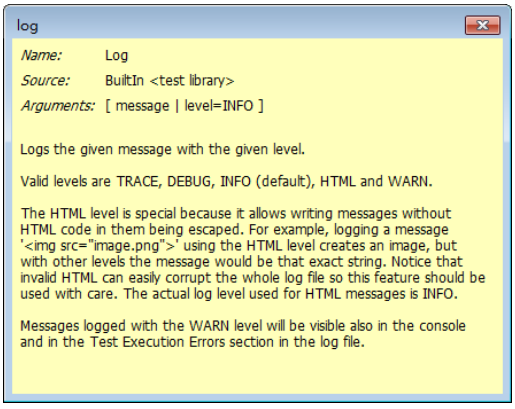

图 2-8-24

它需要 2 个参数，一个是打印的日志 message；另一个是日志的级别 level。level 本身有个默认值，如果不需要改变的话可以不传值。

结合这 2 个关键字来做一个例子，先看一下脚本，见表 2-8-13。

表 2-8-13

Run Keyword	log	abcd	WARN

这是知道它的参数一个一个地写，那么 abcd 和 WARN 其实都是 log 的参数，放到 Run Keyword 里就是*args，那么可以用一个 List 变量来存这 2 个值，脚本见表 2-8-14。

表 2-8-14

@{argVal3}	Create List	abcd	WARN
${keyword}	Set Variable	log	
Run Keyword	${keyword}	@{argVal3}	

这样写就是完全符合了关键字里说的第一个 Scalar 变量，第二个 List 变量了。

运行案例，运行结果如图 2-8-25 所示。

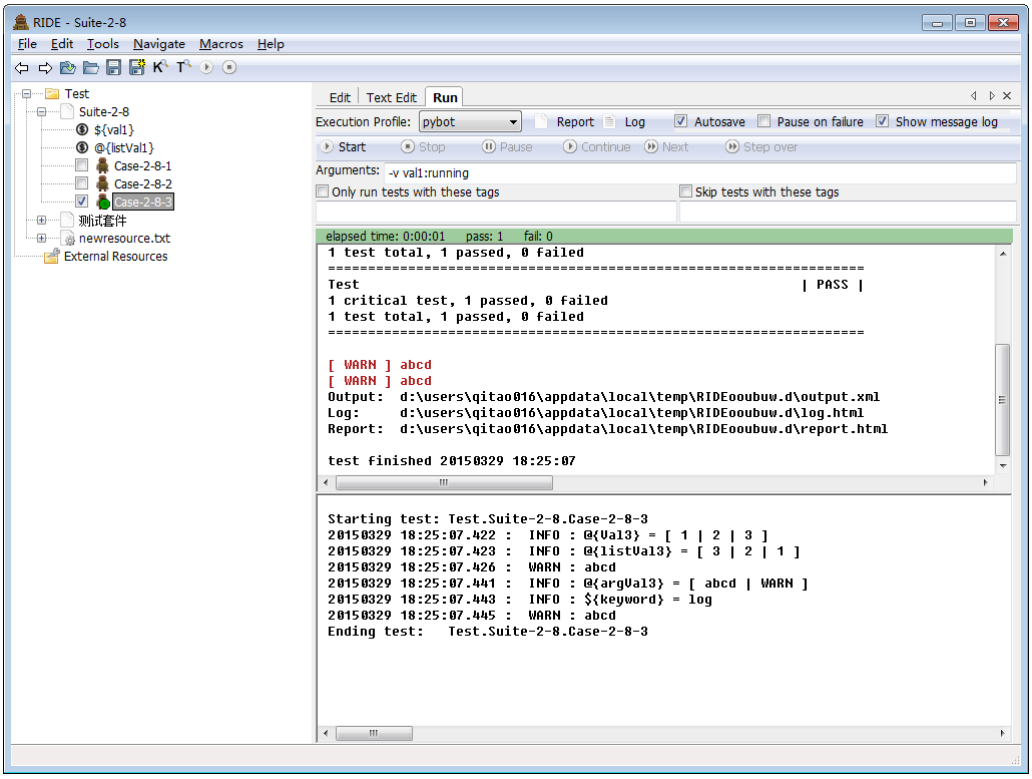

图 2-8-25

和我们预期的一样，两种写法都打出来 WARN 的 abcd 的日志。

当然，并不是说没有星号的就不能用 List 变量，只要是变量个数和 List 变量里元素的个数相同，就可以用 List 传值。比如像 Log 这个关键字，它有 2 个 Scalar 型的传入参数，如果想传 List 也是可以的，但是必须是一个有 2 个元素的 List。另外，因为 Log 的第二个参数有默认值，所以如果传入 1 个元素的 List 也是可以的，但是如果传入 3 个元素的 List，那么就会报错了。

刚才的@{argVal3}刚好就是 2 个元素的 List，也正好是 log 需要的参数，所以可以加一行 Log 的脚本，如图 2-8-26 所示。

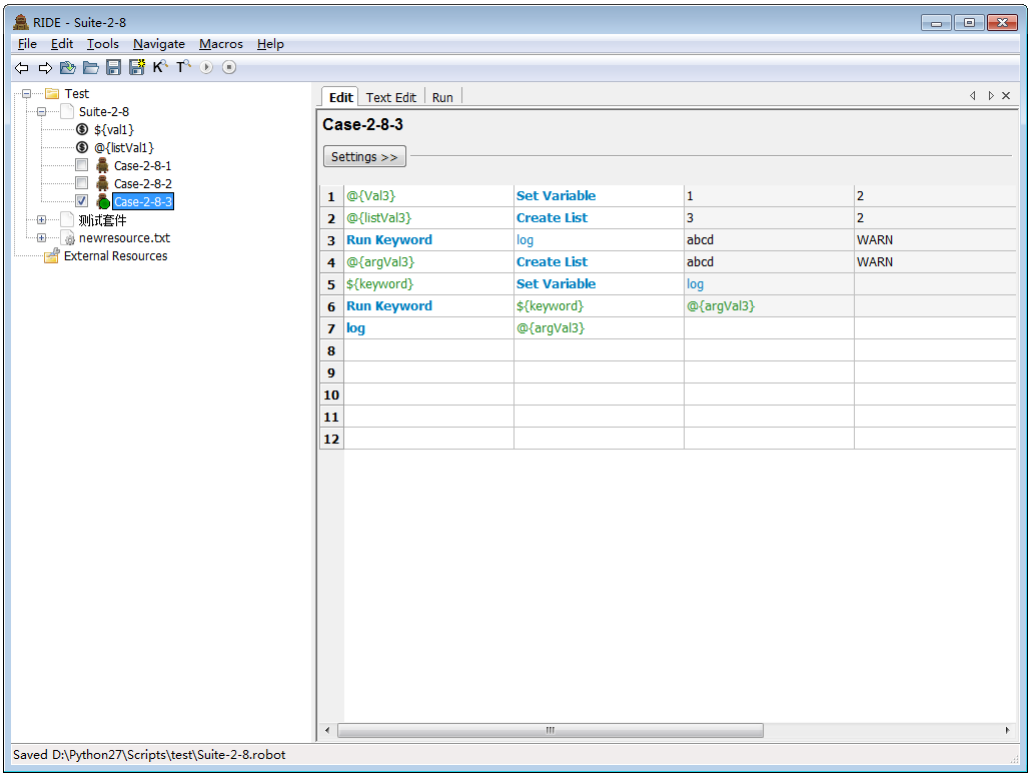

图 2-8-26

运行后的结果也是一样的，如图 2-8-27 所示。

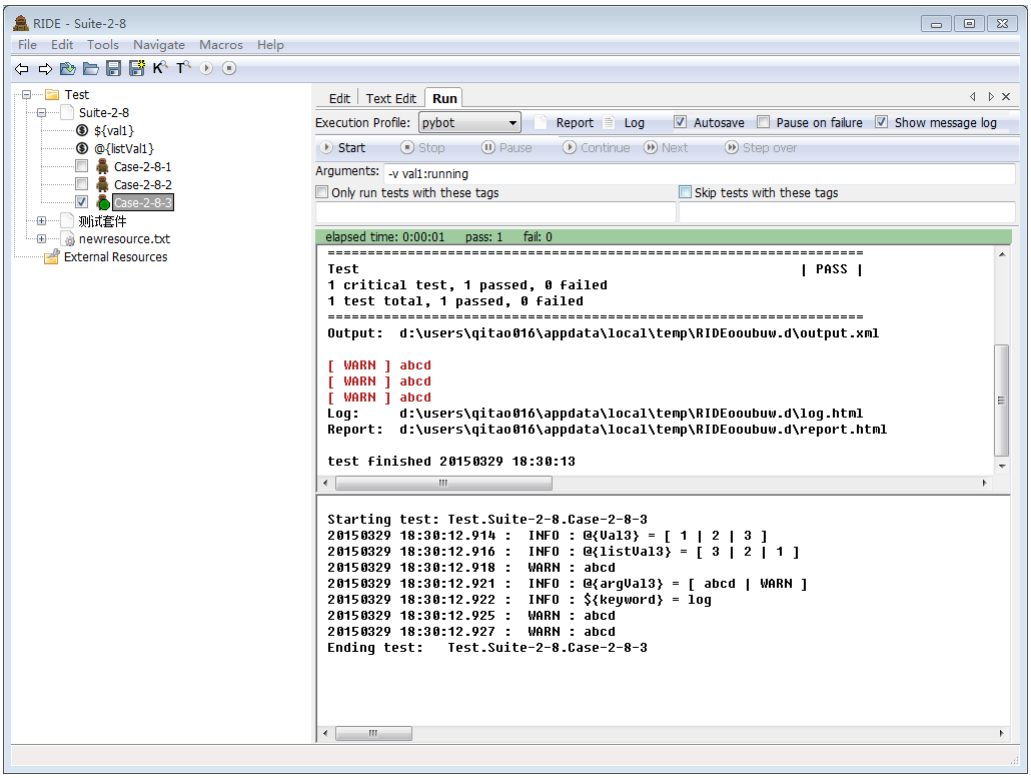

图 2-8-27

综上，对于有多个参数的关键字，你可以按照需要传入多个 Scalar 变量，也可以传入 List 变量，但是 List 的元素个数必须要满足关键字需要的最少参数个数，根据你自己的需要选择合适的方式。

3. List 元素的使用

前面一直在介绍整个 List 变量，但是其实更多的时候是使用 List 变量里的元素，这些元素的值的获取有两种方式，正常的方式是“@{变量名}[index]”，在 List 的变量的括号外面加上方括号，里面是索引值。

另一种方式是“${变量名[index]}”，就是用“$”的变量，在变量名后面加上方括号，里面是索引值。如果是多维 List，那就看需要取哪一维，取单个元素的话，是几维就需要几个“[]”，但是实际使用要注意“[]”的位置。

（1）一维 List

继续在上面的案例里新增一段脚本，见表 2-8-15。

表 2-8-15

@{useList}	Create List	a	b	c
log	@{useList}[1]			
log	${useList[1]}			

这里把另一种形式也打印出来了，运行结果如图 2-8-28 所示。

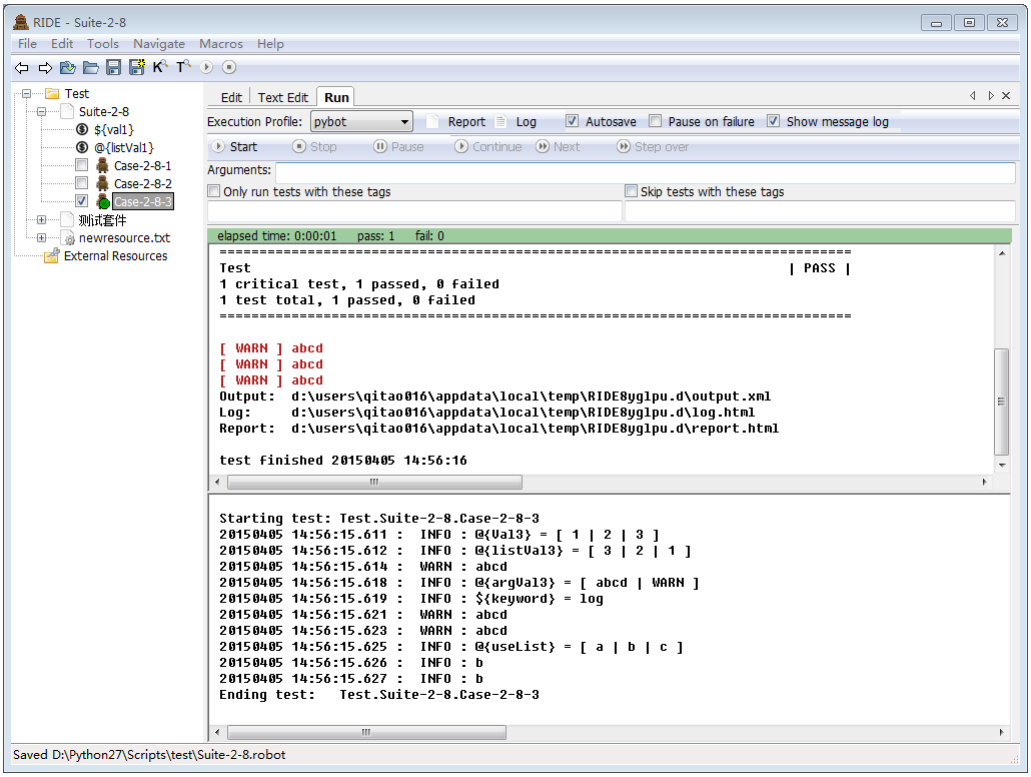

图 2-8-28

（2）二维 List

继续新增脚本，见表 2-8-16。

表 2-8-16

@{listA}	Create List	1	2	
@{listB}	Create List	3	4	
@{listC}	Create List	${listA}	${listB}	5
log	@{listC}[1][1]			
log	${listC[1][1]}			

这里要注意，在给@{listC}赋值时，后面的 listA 和 listB 要用“$”符号，这样才是二维 List，如果写成“@”的话，最后 listC 还是一维的。

另外，这里的“@{listC}[1][1]”的颜色不正确，因为这个写法是有问题的，如图 2-8-29 所示。

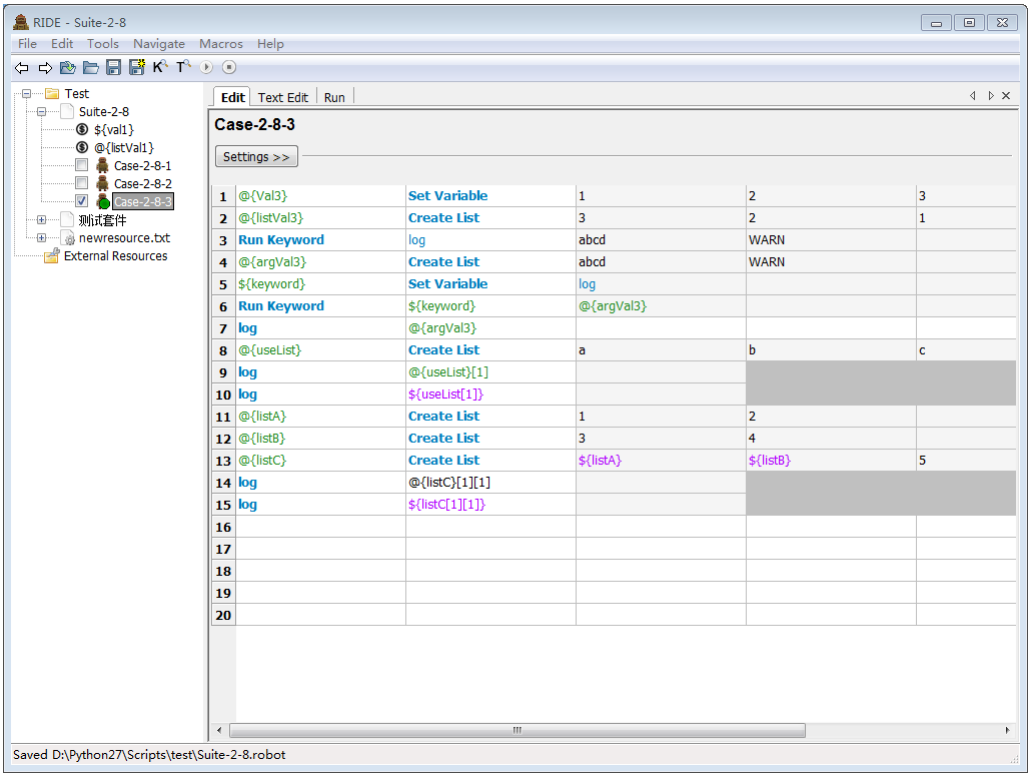

图 2-8-29

运行结果如图 2-8-30 所示。

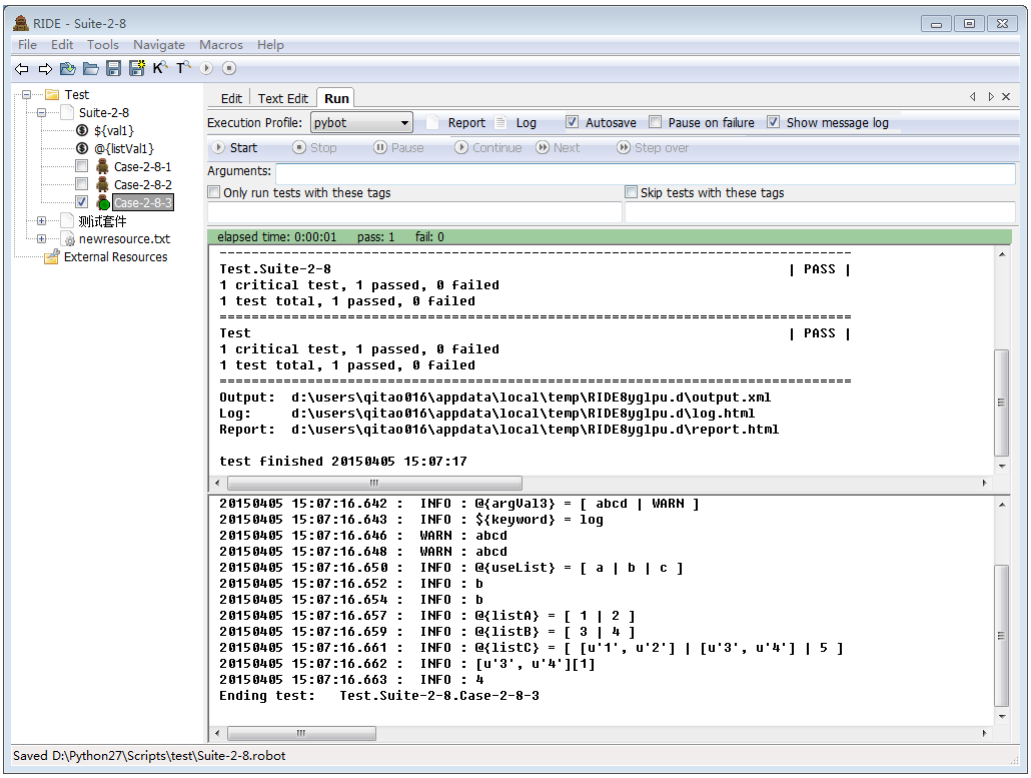

图 2-8-30

可以看到这里只有一个“[1]”生效了，即@{listC}[1]的值是[u'3',u'4']，然后这个值和后面的“[1]”组成了一个字符串连接。

正确的写法要写成@{listC[1]}[1]，加上这个正确写法的变量的脚本，运行的结果如图 2-8-31 所示。

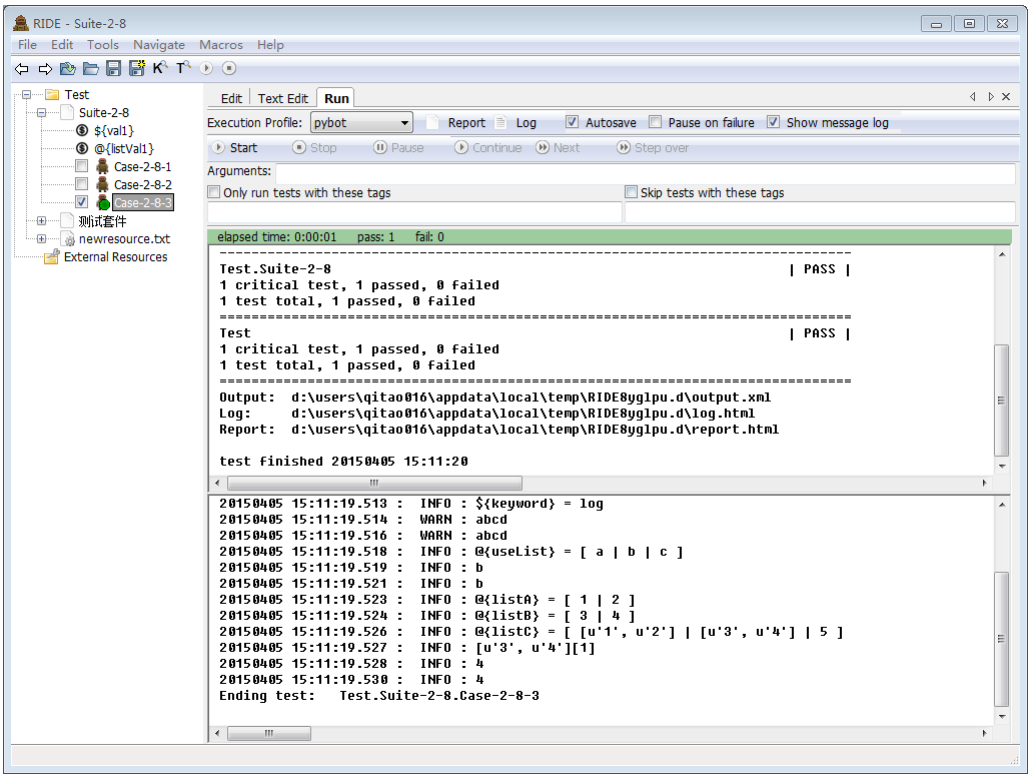

图 2-8-31

总之，就是用方括号来表明需要的元素，如果是多维 List，要想清楚到底要哪层的，不要写少了或者写多了。

2.8.4　变量转换

之前提过了 Scalar 变量用“$”作为标识符，List 型变量用“@”作为标识符，而且它们是可以进行转换的（前面几节里也有一部分转换的例子）。理论上可以互相进行转换，但是也有一点限制。

1. List 变量转换成 Scalar

这种转换的效果是把整个 List 变成一个 Scalar，笔者觉得作用主要是对于那些只接收 Scalar 变量参数的关键字，而又想传 List 的全部值时，或者其他你需要的把 List 当作 Scalar 的情况下使用。在前面一节里做过 Log 的例子，它需要 2 个 Scalar 变量的参数，也可以用一个 2 个元素的 List 来传值给 Log，但是那是因为 2 个元素分别对应 Log 的 2 个参数。如

果想要用 Log 打印出来 List 的 2 个元素，那就可以直接把@{argVal4}换成${argVal4}，虽然${argVal4}没有初始化过（图中显示为紫色，书上看不出来，后面提到紫色的，请在自己的 RIDE 里打开案例查看），但是因为有了@{argVal4}，那么默认${argVal4}也就相当于初始化过了，值与@{argVal4}相同。脚本如图 2-8-32 所示。

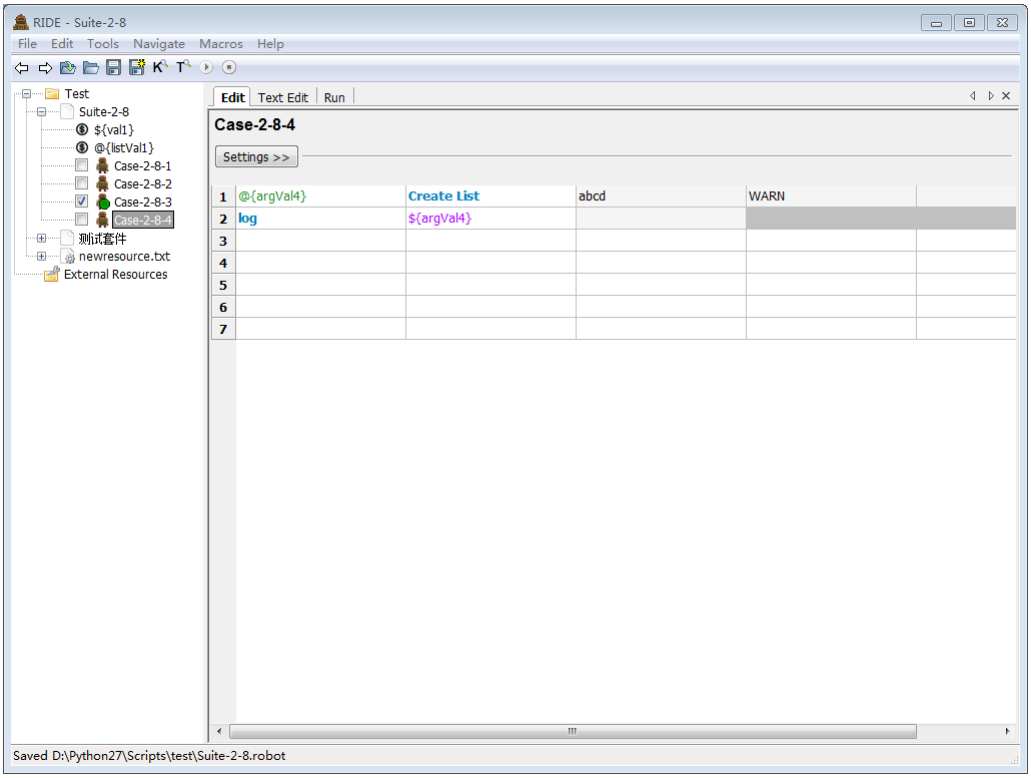

图 2-8-32

运行结果如图 2-8-33 所示。

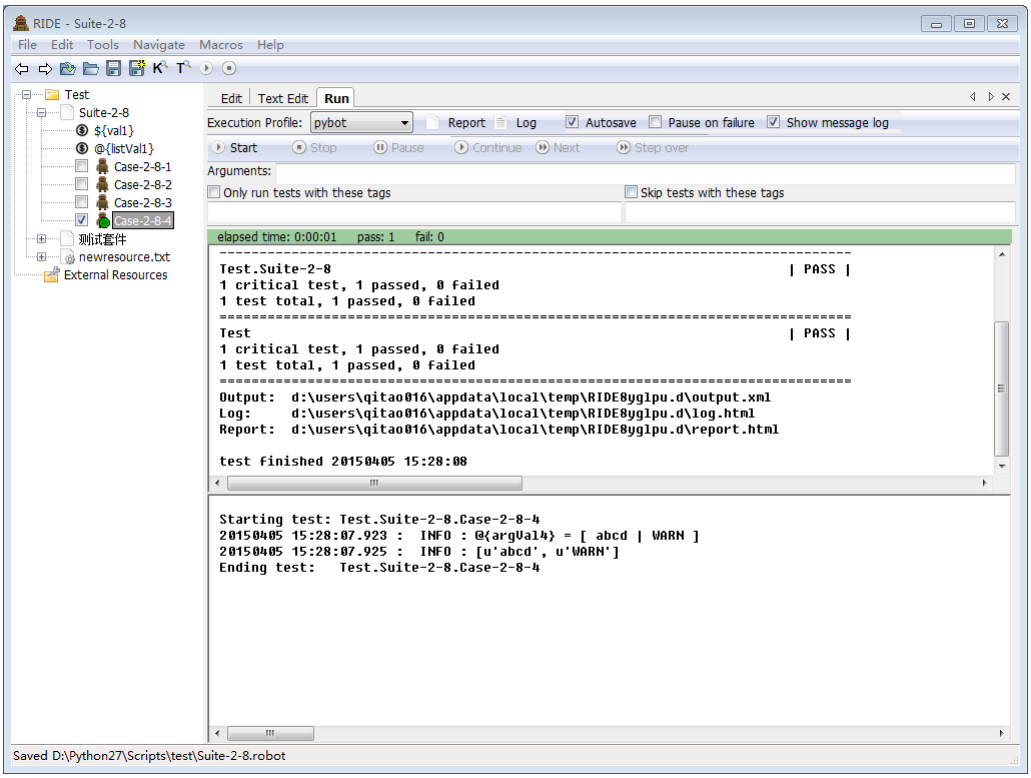

图 2-8-33

这里就是把@{argVal4}转换成了${argVal4}。

2. Scalar 变量转换成 List

其实以前介绍用户关键字的时候也演示过，对于返回值是 List 的关键字，如果给要赋值的变量写的是 Scalar 的，它会自动把它变成 List 的。

还是上面这个例子，新增的脚本里改一下，将 Create List 前面的@{argVal4}换成${argVal4}，这样也是可以的，运行结果如图 2-8-34 所示。

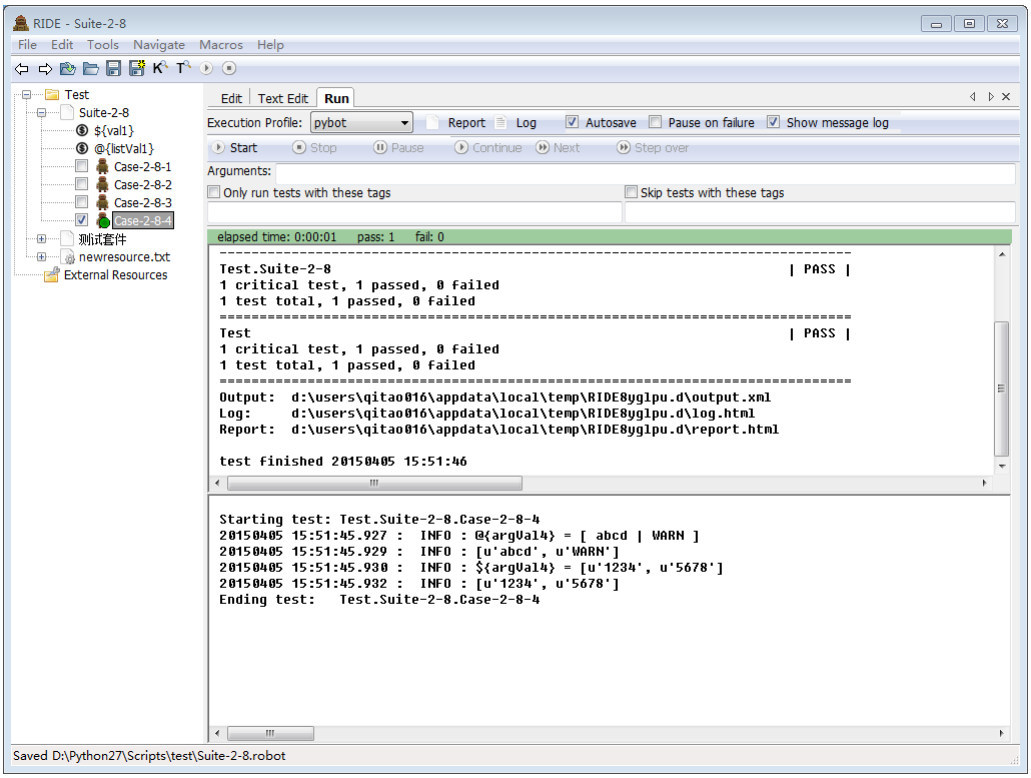

图 2-8-34

这里的${argVal4}已经是一个List值的变量了。同样也可以把Log后面的变量换成“@”的，这里换个变量名 listVal4，避免和前面的冲突。同时由于 Log 后面只要 Scalar 的值，加点其他字符让他们组成字符串，见表 2-8-17。

表 2-8-17

${listVal4}	Create List	1234	5678
log	=@{listVal4}=		

同样@{listVal4}没有初始化过，在图中也是紫色的，因为初始化了${listVal4}，所以@{listVal4}也相当于初始化了，运行一下，结果如图 2-8-35 所示。

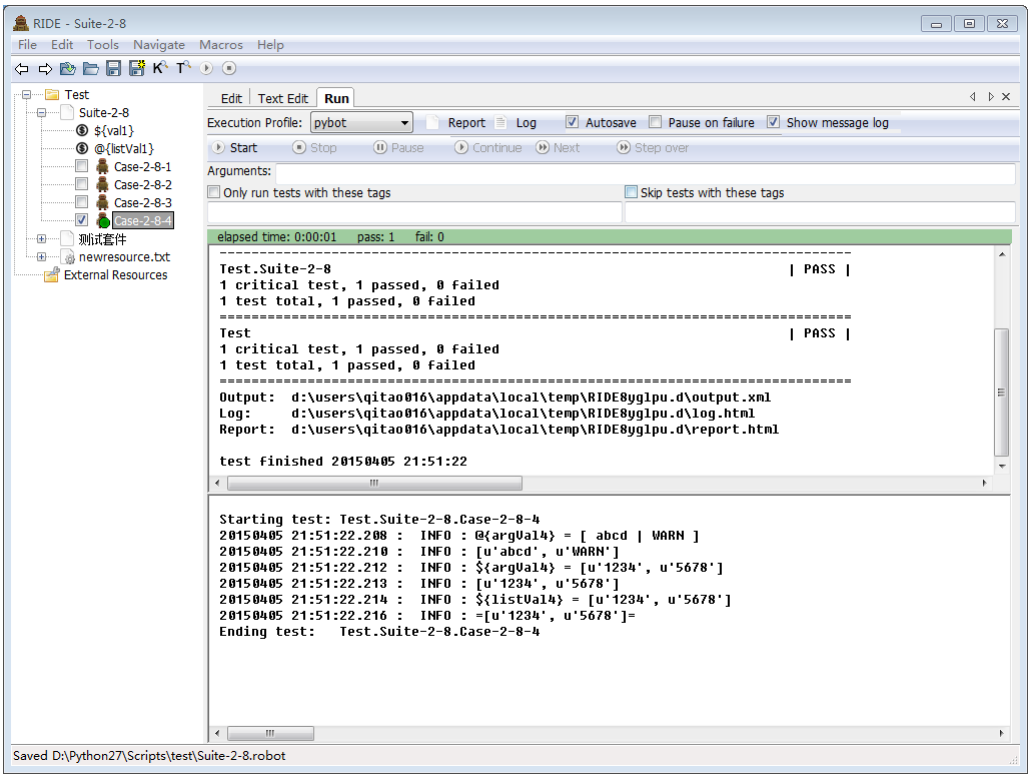

图 2-8-35

这样就相当于把${listVal4}转换成了@{listVal4}。

3. 转换的限制

前面把@{argVal4}转换成了${argVal4}，也把${listVal4}转换成了@{listVal4}，为什么要换个变量呢？这里就是限制了。

在前面的小节里提到，变量要进行赋值（类似于初始化）之后才能使用，如果没有进行过赋值的，在 RIDE 里会是“紫色”提示。

系统会默认${argVal4}是@{argVal4}的 Scalar 形式，也会默认@{argVal4}是${argVal4}的 List 形式。这里的限制就是变量必须没有初始化或者赋值过，才能这样“呼唤”。如果初始化或赋值了，它们就各自独立了。

先打印一下@{argVal4}的值，然后再把第一次从@{argVal4}转换成了${argVal4}的脚本复制一份，放到后面，顺便修改一下值，脚本见表 2-8-18。

表 2-8-18

log	=@{argVal4}=		
@{argVal4}	Create List	4444	8888
log	${argVal4}		

因为这个脚本是跟着前面的案例一起的，那么大家不妨猜一下${argVal4}的值会是[u'4444',u'8888']么？先运行一下，看看结果再分析，如图 2-8-36 所示。

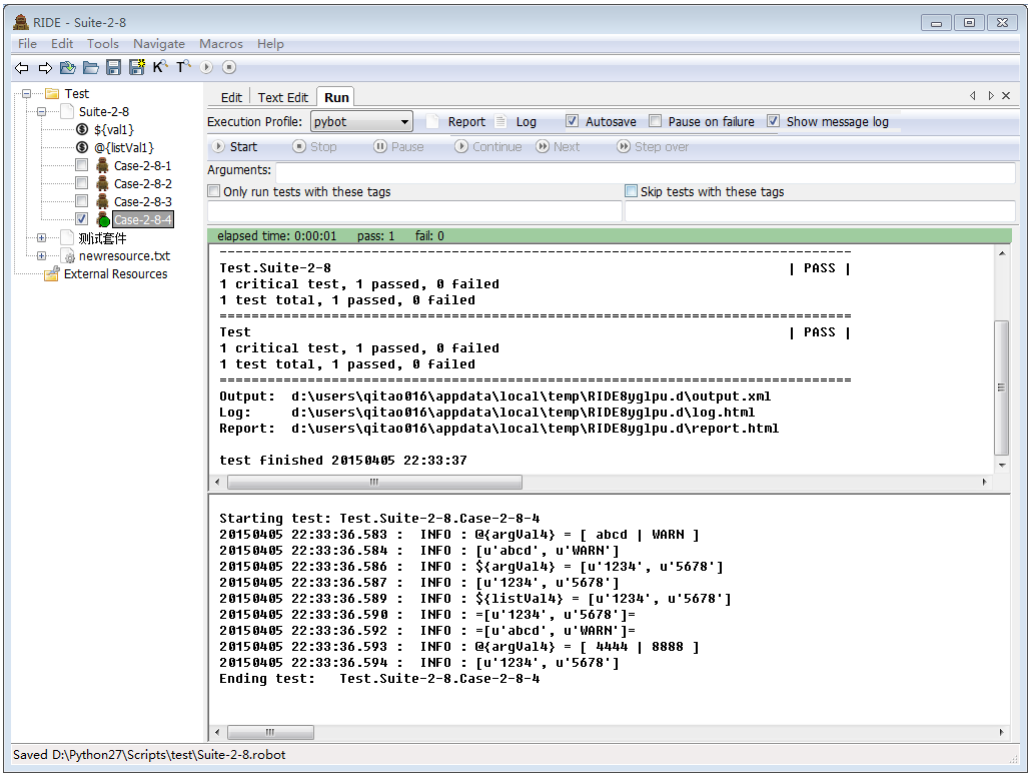

图 2-8-36

可以看到@{argVal4}的值已经变成[4444 | 8888]了，但是${argVal4}的值还是[u'1234',u'5678']。再看一下案例，如图 2-8-37 所示。

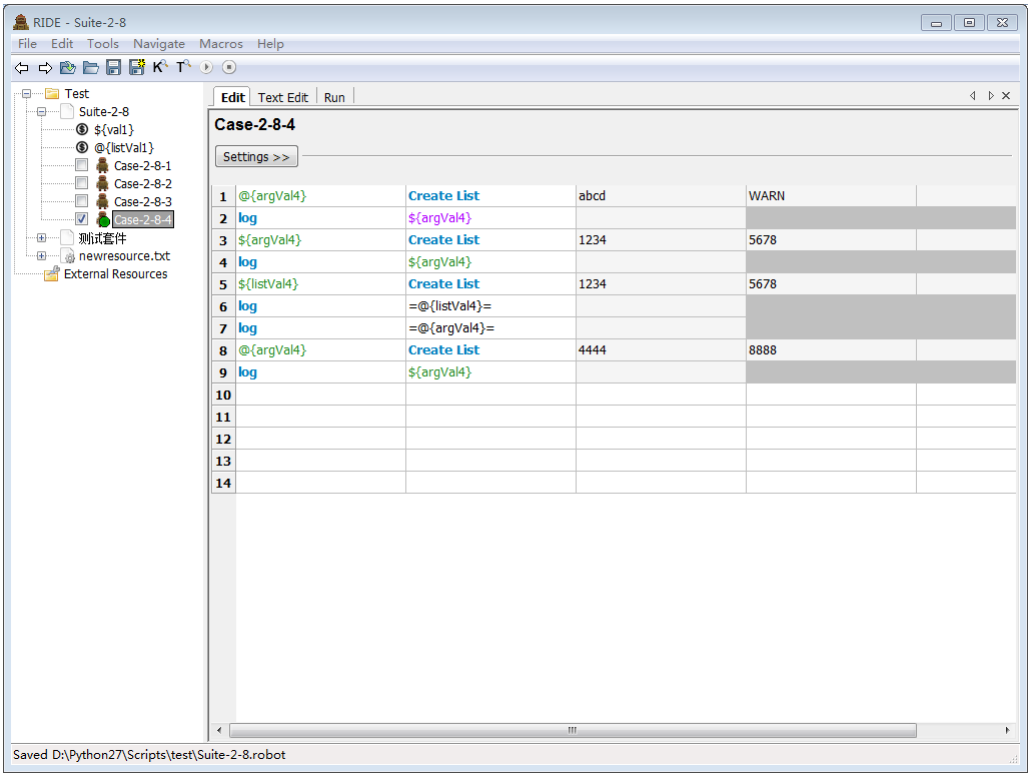

图 2-8-37

其实在第 1 行，就已经给@{argVal4}赋值了[u'abcd',u'WARN']，此时没有声明${argVal4}，所以在第 2 行 log ${argVal4}的时候，它的值还是@{argVal4}的值。在第 3 行又给${argVal4}赋值了[u'1234',u'5678']，此时@{argVal4}和${argVal4}已经分别进行了初始化，它们之间就不能再进行互换了。因此能看到第 7 行的@{argVal4}还是[u'abcd',u'WARN']，而在第 8 行把@{argVal4}重新赋值为[4444 | 8888]，但是这样也不会影响${argVal4}的值。

是不是有点绕，总结概括一下就是，如果一个变量要转换成另一类变量，只要目标变量没有初始化或者赋值过，就可以进行转换。如果目标变量已经初始化或者赋值过了，那就无法进行转换了。所以大家以后在定变量名字时注意这一点，不要让这样的限制影响到你的案例设计。

2.8.5 其他变量

其实除了 Scalar 变量和 List 变量，在 Robot Framework 里还有 Dictionary 变量和对象变量，它们的展现形式其实都是类似于 Scalar 变量，这也是为什么笔者在前面的小节里说 Scalar 变量要取决于它的值。值不同，它就代表了不同的变量，像前面的转换里的${argVal4}的值就是 List 的。

这里简单介绍一下 Dictionary 变量和对象变量，因为后续会有合适的章节来展示这些变量的定义和使用。

Dictionary 变量：Dictionary 其实和 List 有点类似，List 的每个元素是一个值，Dictionary 的每个元素是一对值，比如{key1:val1,key2:val2}，这就是一个 Dictionary，它的元素都是成对的，一个 key 对应一个 Value，key 是不能重复的。

对象变量：对象变量将在接口测试中的 Request 测试库里接触到，假设${return}是接口返回的对象。对于对象来说，更多使用的是对象的属性值。比如这个对象的状态就是${return.STATUS}，对象的内容就是${return.CONTENT}，具体的例子在后面的章节里介绍。

2.8.6 变量文件

前面介绍过怎么添加 Library，怎么添加 Resource，其实在那里还有一个 Variable 变量文件可以添加，变量文件只支持 Python 的 py 文件，这里有一个官方的例子，可以新建一个 var.py 文件，里面输入如下内容：

```
import random
__all__ = [`scalar`, `LIST__var`, `integer`]
scalar = `Hello world!`
LIST__var = [`Hello`, `list`, `world`]
integer = random.randint(1,10)
```

有几点要注意的，一个是“__all__”，这是用来列出这里都有哪几个变量的；另一个是“LIST__var”，它真正的变量名是 var，只是用“LIST__”来标明它是一个 List 变量，其他 2 个都是 Scalar 变量。

可以在 Suite 的 Import 的地方去引入这个文件。还有一个方法是在运行时，通过“-V”的参数来引入这个文件，这里是大写的 V，要注意和之前小写的 v 是不一样的。

2.9　Keyword 关键字

Robot Framework 主要就是关键字驱动的自动化测试，关键字是它的核心。从关键字的类型来说，笔者更愿意分为系统关键字和用户关键字两种，系统关键字通常都是来源于测试库，用户关键字更多的是来源于资源文件（当然你也可以在测试套件里加用户关键字，但是不推荐），而自己可以创建的是 User Keyword 用户关键字。其实系统关键字底层就是 Python 的函数，而用户关键字其实和函数也没什么两样，读者完全可以像设计函数一样设计你的用户关键字。

前面说到了可以在资源文件里添加用户关键字，也可以在测试套件里添加用户关键字，后者不推荐。所以，这里主要介绍资源文件里添加用户关键字，想在测试套件里添加用户关键字的读者，可以自行尝试。

这一章将新建一个 Suite-2-9 来存放本章的案例，以后就不再提示了。

2.9.1　用户关键字

1. 新增用户关键字

首先在之前的资源文件 newresource.txt 上单击鼠标右键，选择“New User Keyword”命令，如图 2-9-1 所示。

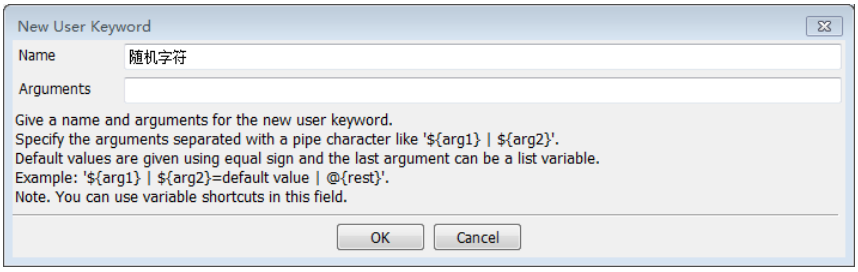

图 2-9-1

在“Name”文本框中输入名称，中文英文都可以，这里输入 Name 为“随机字符”，Arguments 可先不管，单击“OK”按钮，如图 2-9-2 所示。

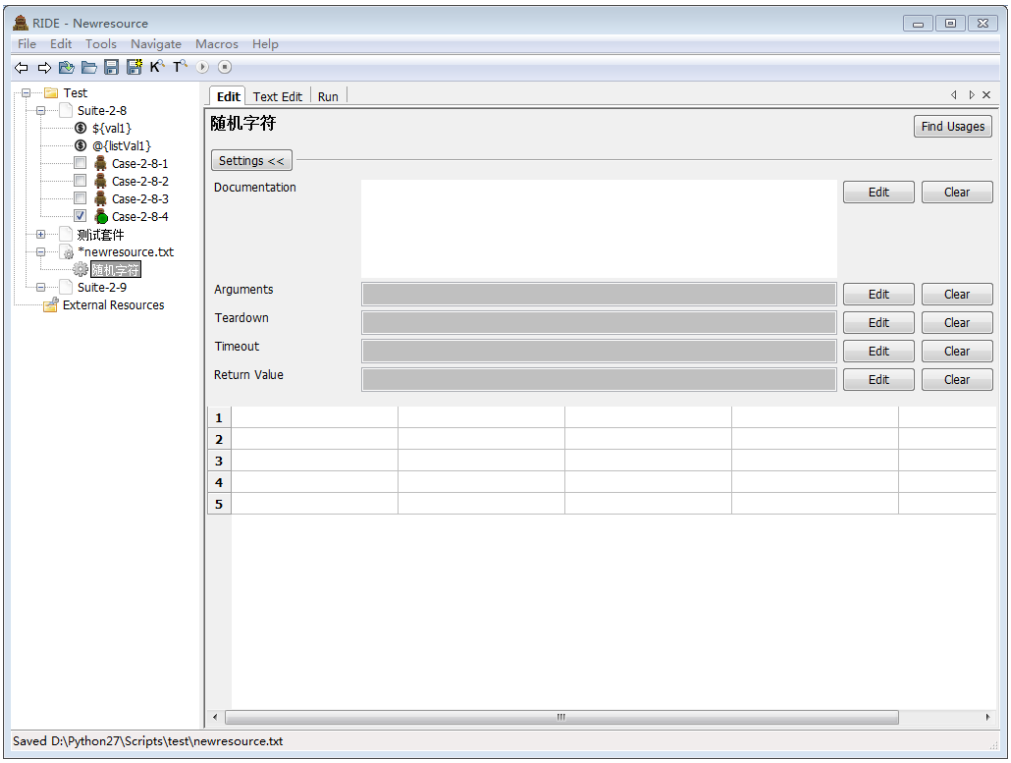

图 2-9-2

2. 快捷菜单

在 User Keyword 里可以做什么？在“随机字符”上单击鼠标右键，如图 2-9-3 所示。

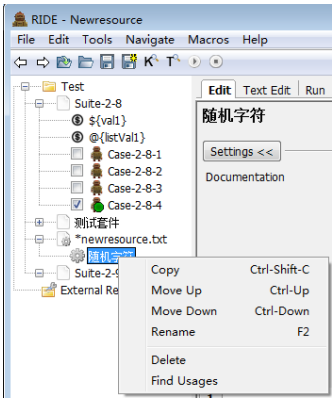

图 2-9-3

可以看到和 Test Case 里基本一样，就是多了一个“Find Usages”命令，这个和工作区 Edit 界面里的 Find Usages 作用是一样的。

Find Usages：可以找到这个关键字在哪里使用了，也可以通过单击界面上的“Go to definition”命令返回到定义这个关键字的地方。

3. Settings 设置项

再看一下 Settings 设置项里的内容，和 Test Case 有些区别，如图 2-9-4 所示。

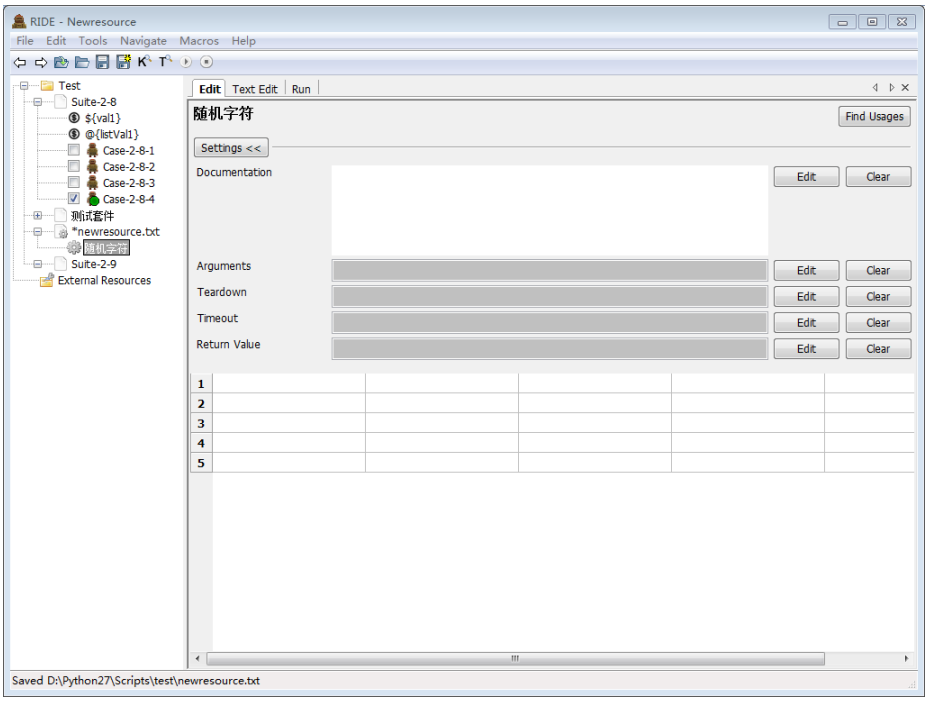

图 2-9-4

- Documentation：文档；
- Arguments：关键字的传入参数；
- Teardown：和 Case 的类似，设置关键字执行完成时的动作；
- Timeout：和 Case 的类似，运行超时的时间设置；
- Return Value：返回值。

除了 Teardown 和 Timeout 是测试案例里也有的，多出来的有传入参数和返回值，有了它们，用户关键字才是真正的“如鱼得水”。

2.9.2 传入参数 Arguments

在前面看过了一些系统关键字的参数，比如 Log 和 Set Variable。再看一下它们的 Arguments，如图 2-9-5 和图 2-9-6 所示。

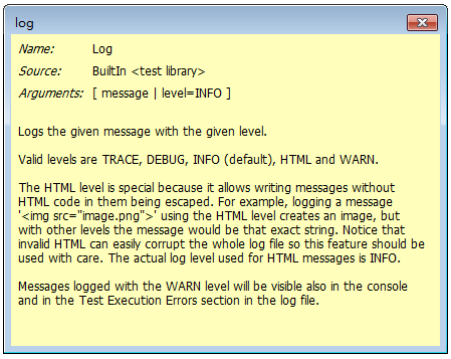

图 2-9-5

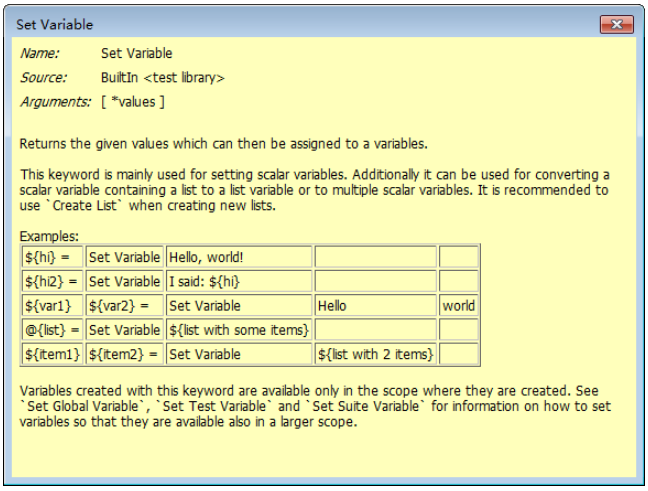

图 2-9-6

从这两个关键字的参数可以看到，常用的参数其实和变量也是相关的，主要就是 Scalar 变量和 List 变量。而参数分类来说，一般就是 3 种，必填参数、非必填参数和 List 参数，下面一一来说。

（1）必填参数

Log 关键字里的第一个参数 message 就是必填参数，只要是这种参数，后面没有默认

值的就是必填参数。

先给“随机字符”加 2 个参数${arg1}和${arg2}，参数之间使用“|”进行分隔，如图 2-9-7 所示。

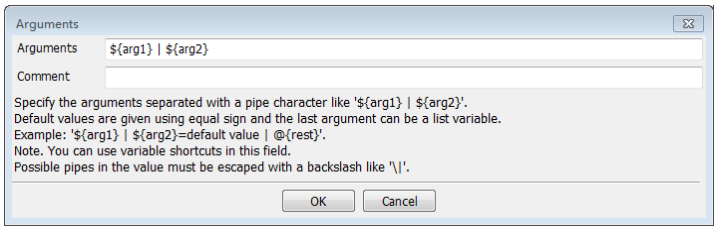

图 2-9-7

这样添加的参数，表示这些参数都是必填的，建一个 Case-2-9-2，并调用这个关键字测试一下（调用之前需要先在 Suite 里加载资源，大家参考前面的来加）。这里的随机字符后面的 2 个格子都是红色的，提示参数必填。书上无法看出颜色，请跟着例子做，然后看一下是不是红色，如图 2-9-8 所示。

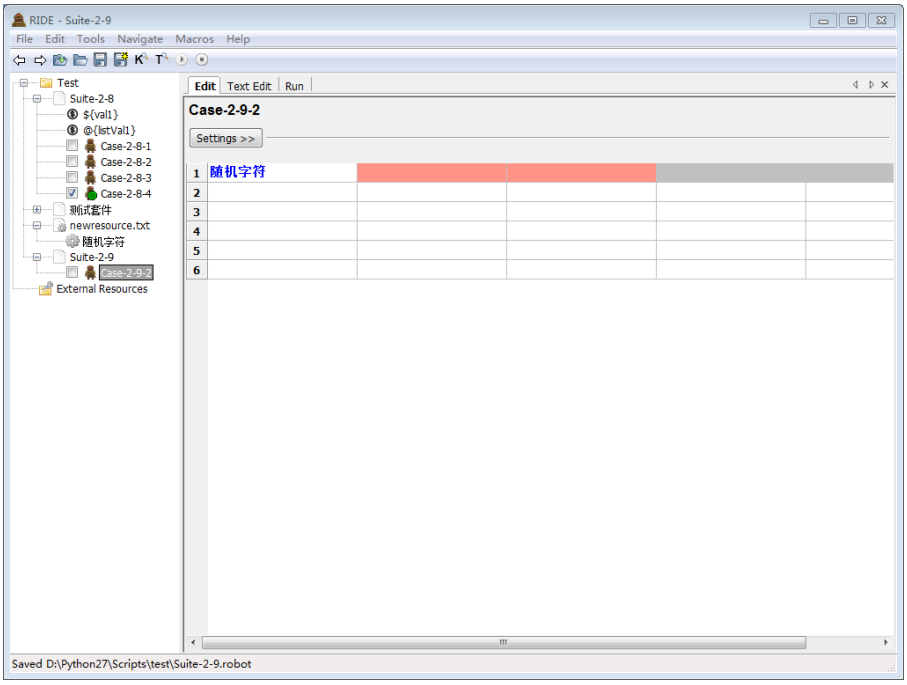

图 2-9-8

把鼠标放到“随即字符”上，按 Ctrl 键，可以看到此时关键字的说明，如图 2-9-9 所示。

图 2-9-9

（2）选填参数

Log 关键字里的第二个参数 level=INFO 就是选填参数，主要是因为它有一个参数的默认值，默认值是用“=”加上值来设置，如果想默认为空，只写等号就可以了。

默认值的意思就是，如果在调用这个关键字时，如果不给这个选填参数传值，那么就使用默认值作为参数的值。

tips：因为参数之间使用“|”进行分隔，所以如果想在默认值里使用“|”作为一个值，而不是分隔符，那么就要使用“\|”来表示“|”的值。

回到关键字“随机字符”的界面，在 Arguments 上单击一下鼠标（或者单击“Edit”按钮），先试一下把第一个参数加上默认值，如图 2-9-10 所示。

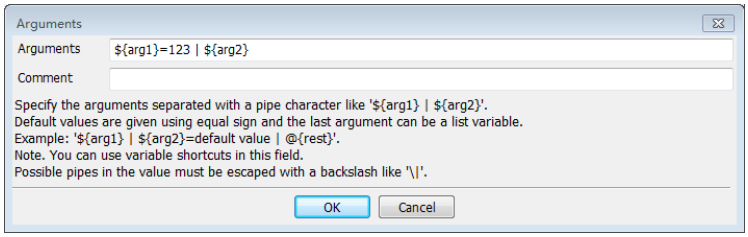

图 2-9-10

如果直接这样加的话，单击“OK”按钮后，会弹出报错对话框，如图 2-9-11 所示。

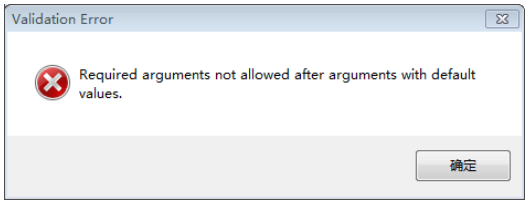

图 2-9-11

此对话框的意思是说，必填参数不允许在选填参数后面。

也就是说，如果某一个参数设置成了选填参数，那么它后面的参数都必须是选填参数，

不能是必填参数。

在这里就是第 1 个参数如果要可选，要么把 2 个参数也都设置上默认值，要么就把这个参数调整到最后去。那么先参照 Log 的参数设置，把默认值给第二个参数，就不会报错了，如图 2-9-12 所示。

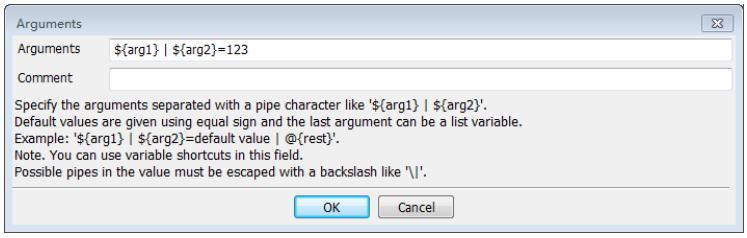

图 2-9-12

再回去看一下案例那里，只有第一个参数是必填的，所以也只有一个红色格子了，如图 2-9-13 所示。

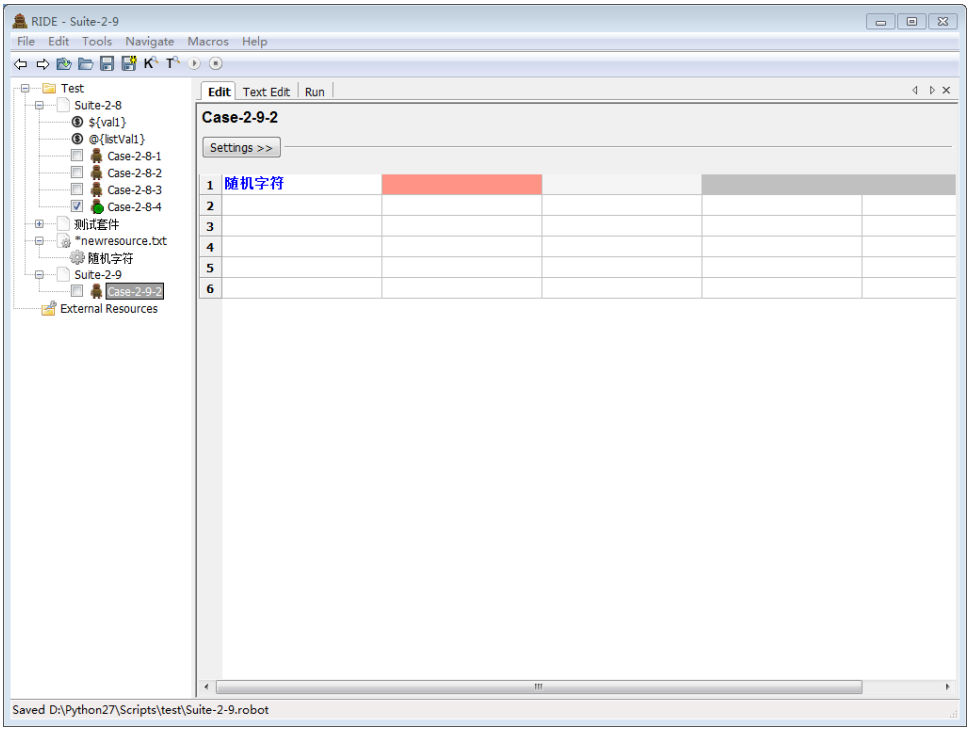

图 2-9-13

（3）List 变量参数

List 变量也是可以作为参数，但是 List 变量只能放在最后一位。如果放在前面，就会报错，比如在${arg1}和${arg2}中间加上一个@{arg3}，如图 2-9-14 所示。

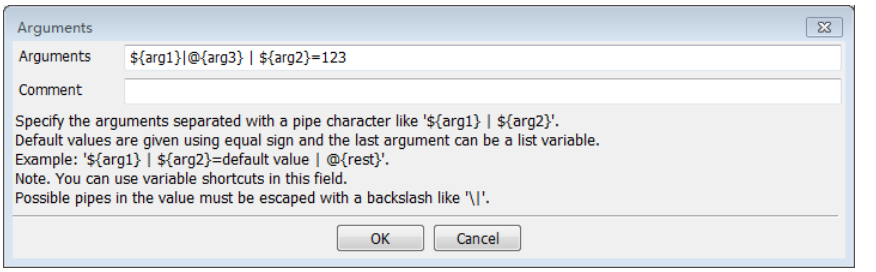

图 2-9-14

保存时就会提示“List variable allowed only as the last argument（List 变量只允许作为最后一个参数）”，如图 2-9-15 所示。

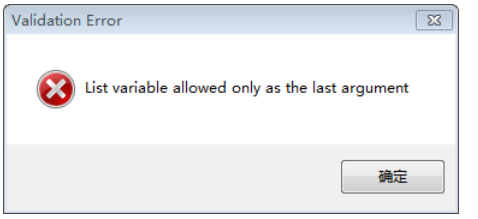

图 2-9-15

为什么只能是最后一个呢？从英文原文来看是“the last argument”，在 argument 后面没有 s。如果一定要试试最后 2 个都放 List 变量呢？也是会报错的（大家可以自己试一下）。

这是为什么呢？首先，List 变量本身来说，它是一个可变的，即 List 的成员数量不确定。而 List 作为参数的话，有几个成员就是相当于几个单值参数，它实际上是提供了一种参数个数可变的方式。那么既然可变，如果放在前面的话（见第一个例子的图），它就没法确定传入的参数到底哪些是给 arg3 的，哪些是给 arg2 的了。同理，如果放 2 个 List 也是不行的，因为也是无法区分传入的参数到底哪些是哪个 List 参数了。

而且 2 个可变和 1 个可变没什么差别，所以最终限定是只能有 1 个 List 参数，并且必须放在最后。

另外对于 List 参数来说，它本身就有默认值了，因为即使是空 List，也是有个空的值

在的，所以对于 List 参数就不用去想着写个“=”加默认值了。

tips：变量名和等号间不能有空格，如果有空格，系统会报参数语法错误。

那先把@{arg3}加到最后，如图 2-9-16 所示。

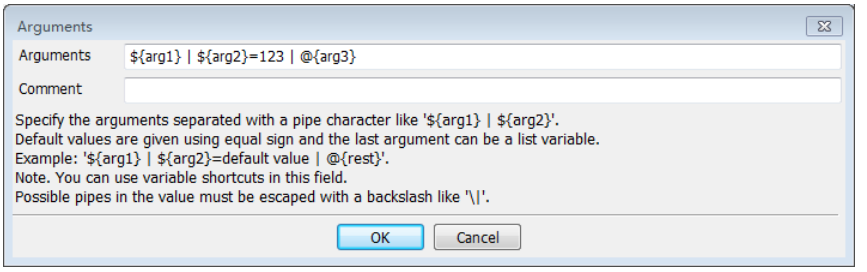

图 2-9-16

最后看一下加上 List 参数后，在 Case 里的关键字说明，如图 2-9-17 所示。

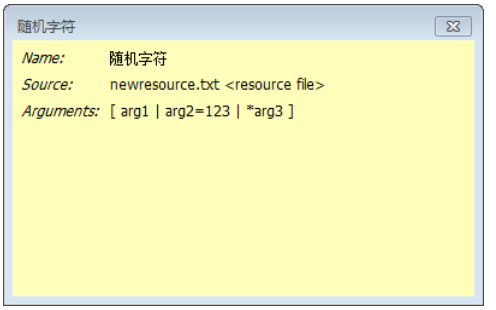

图 2-9-17

可以和前面的 Log 和 Set Variable 关键字的参数对照一下，“*”的就是 List 变量参数。这样 3 种参数都有了，但是关键字里的处理逻辑还没有写，先简单地写一下打印这 3 个变量值的脚本，见表 2-9-1。

表 2-9-1

log	${arg1}	
log	${arg2}	
log	=@{arg3}=	

案例如图 2-9-18 所示。

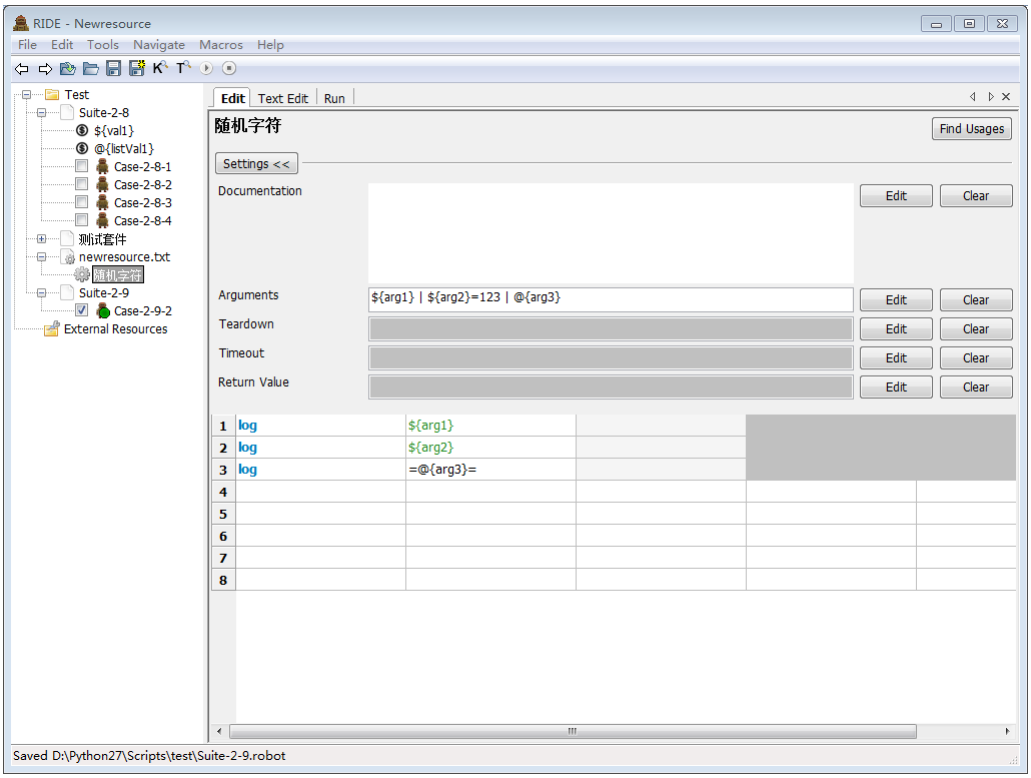

图 2-9-18

然后在 Case 里给它传几个值，见表 2-9-2。

表 2-9-2

随机字符	arg1value		arg31	arg32

笔者跳过了${arg2}的传值，正好看一下默认值的使用。运行案例，看一下打印出来的内容，如图 2-9-19 所示。

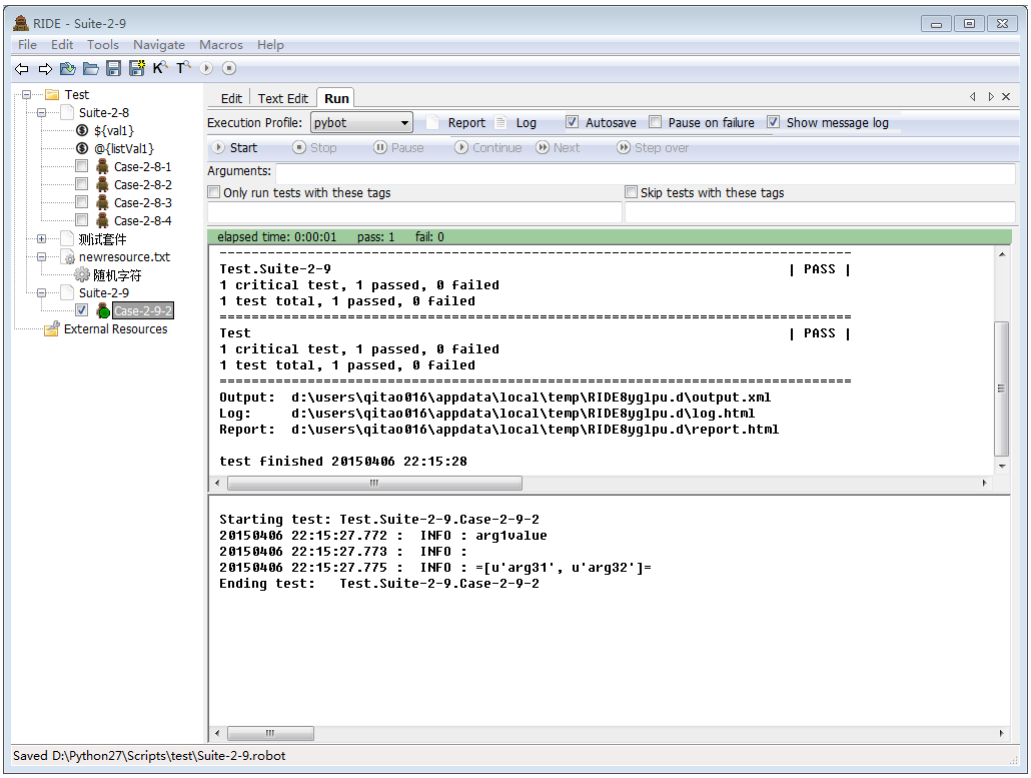

图 2-9-19

看到奇怪的事情了么？${arg2}没有传值，应该用默认值的，但是这里打印出来的是空值。这个原因主要是因为给@{arg3}传了值，虽然空出来，没写${arg2}的值，但是这时的 Robot Framework 就会认为传的真的是空值。那么怎么样才能用上默认值呢？那就是在后面的@{arg3}都不要传值，见表 2-9-3。

表 2-9-3

随机字符	arg1value		

运行一下看看结果，如图 2-9-20 所示。

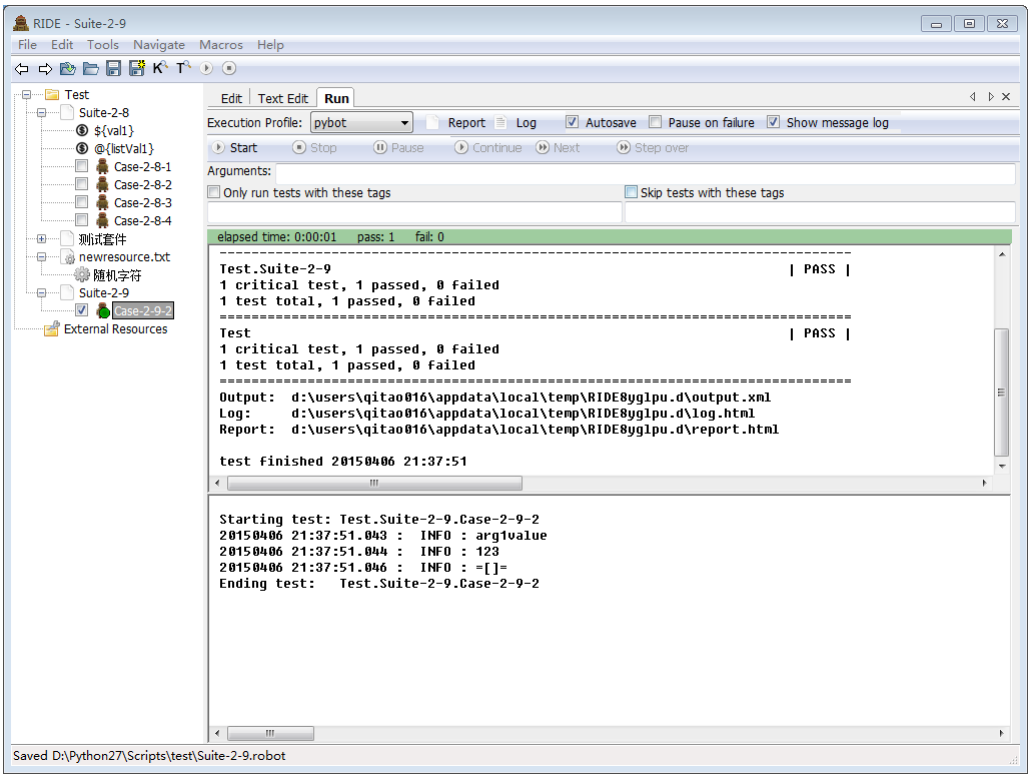

图 2-9-20

这回${arg2}的默认值用上了。这里说明，如果有多个有默认值的参数，如果后面的参数传值了，那么前面的参数也认为传值了，即使你什么都不写，它也认为是传了空值。

2.9.3 Return Value 返回值

返回值这个应该不用怎么解释了，只是说说用法，因为传入参数的时候可以用单值变量，也可以用 List 变量，那么在 Return Value 的时候也是可以用单值变量和 List 变量的。

下面把之前的关键字“随机字符”复制 3 份，分别命名，加上“-1”、“-2”、“-3”，用这 3 个新关键字来举例说明。

（1）单个单值变量返回

在“随机字符-1”里把${arg1}放到 Return Value 里，如图 2-9-21 所示。

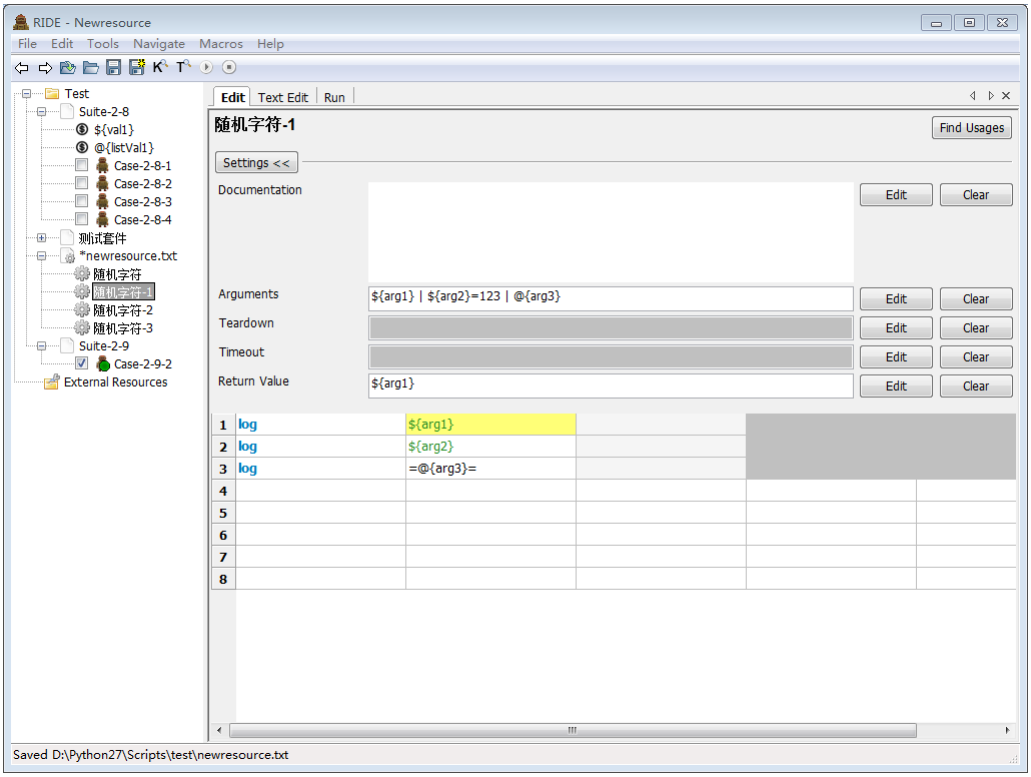

图 2-9-21

在 Case-2-9-3 里，用一个变量${getArg1}获取这个返回值，然后打印出来，脚本见表 2-9-4。

表 2-9-4

${getArg1}	随机字符-1	arg1value
log	${getArg1}	

运行之后，看一下日志打印的效果，如图 2-9-22 所示。

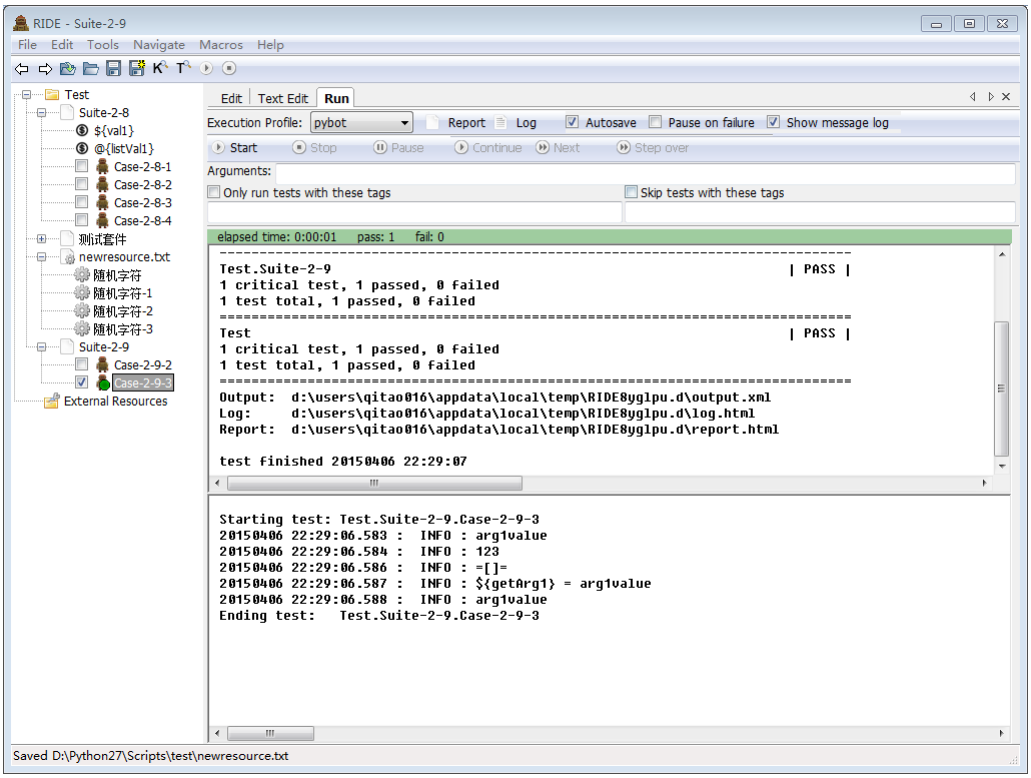

图 2-9-22

（2）多个单值变量返回

在设置返回值时可能大家已经看到了，这里是支持多个变量返回的，也是用“|”分隔的。在“随机字符-2”里把${arg1}和${arg2}都放到 Return Value 里，如图 2-9-23 所示。

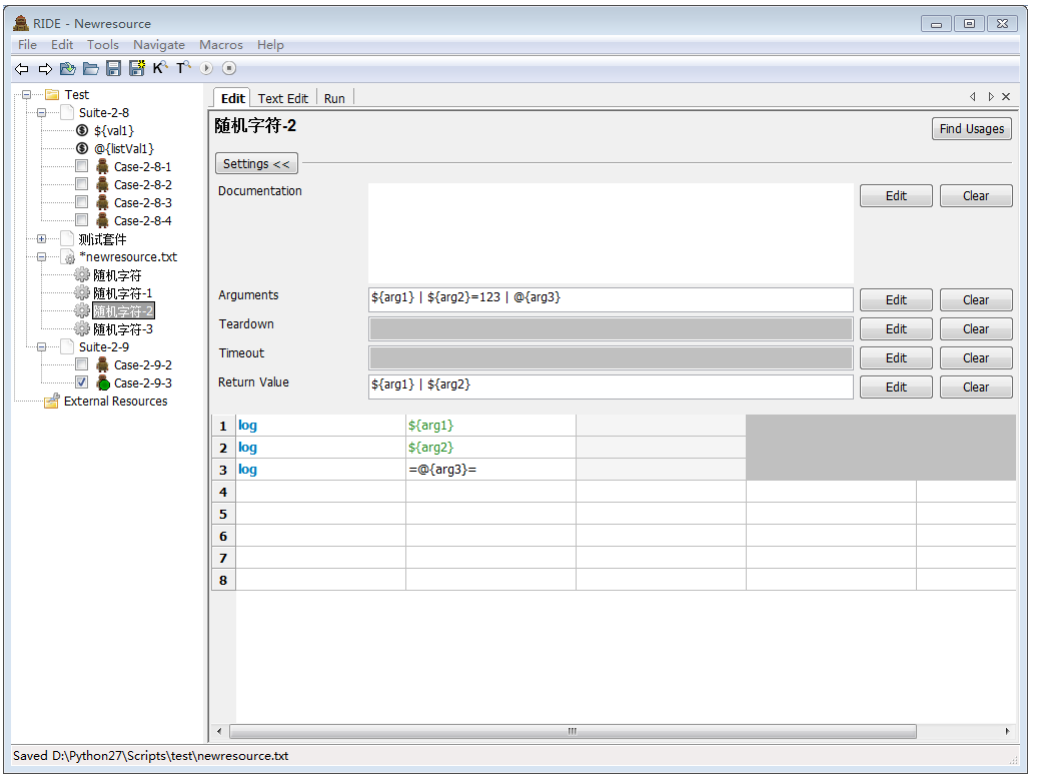

图 2-9-23

1）使用 Scalar 变量接收返回值。

在 Case-2-9-3 里，用一个变量${getArg2}获取这个返回值，然后打印出来。照搬一下前一段脚本，把${getArg1}改成${getArg2}，其他的代码都不动，新加脚本见表 2-9-5。

表 2-9-5

${getArg2}	随机字符-1	arg1value
log	${getArg2}	

前面讲过变量转换的内容，这时${getArg2}转换成了 List，如图 2-9-24 所示。

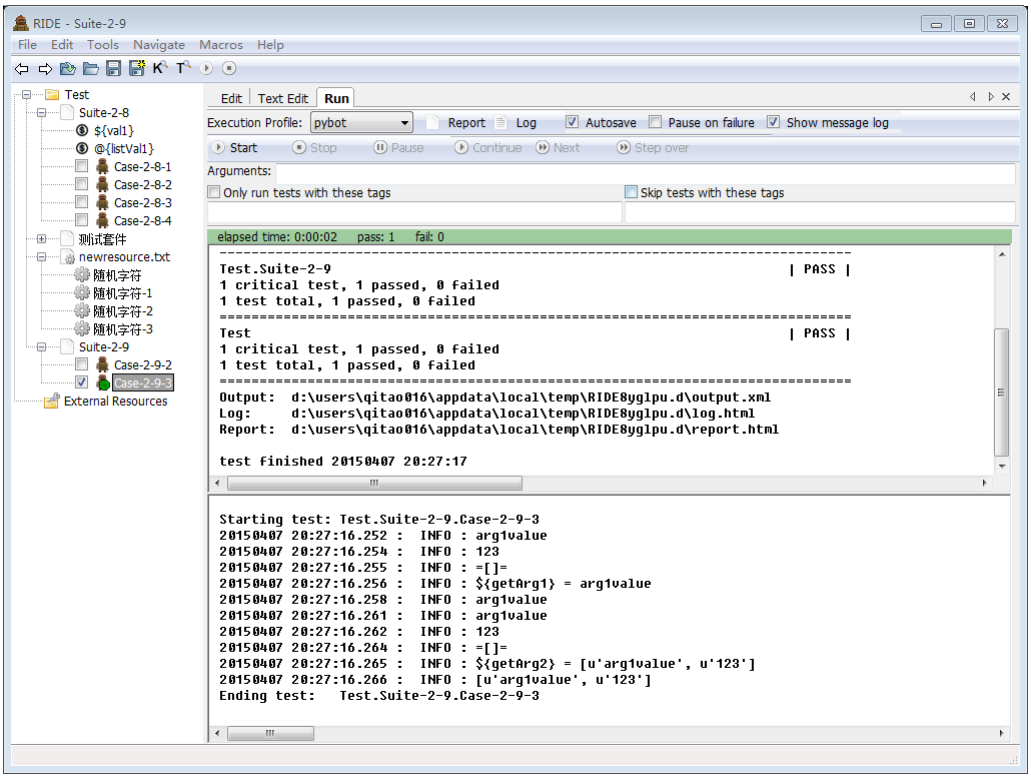

图 2-9-24

2）使用 List 变量接收返回值

因为这里返回了 2 个元素，也知道这里返回的其实是 List，最好用 List 变量来接收，那么在案例中加一个@{ListArg2}获取这个返回值，然后打印出来。为了方便打印，笔者在@{ListArg2}前后加了“=”符号，新加脚本见表 2-9-6。

表 2-9-6

@{ListArg2}	随机字符-2	arg1value
log	=@{ListArg2}=	

再运行一下看看结果，如图 2-9-25 所示。

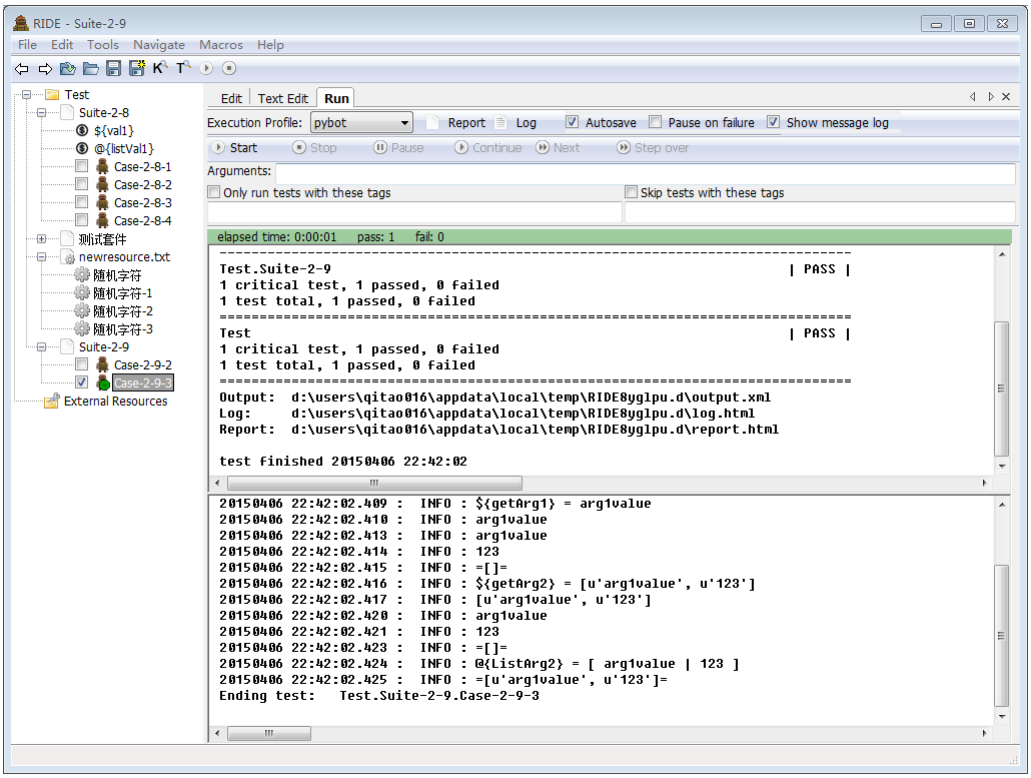

图 2-9-25

3）使用多个 Scalar 变量接收返回值

那么还有一个方式，因为知道有 2 个返回值，所以可以用 2 个变量来获取值，新加脚本见表 2-9-7。

表 2-9-7

${valArg1}	${valArg2}	随机字符-2	arg1value
log	${valArg1}=${valArg2}		

运行一下看看日志，如图 2-9-26 所示。

这样可以直接使用对应的变量了。

上面的 3 种方式，前两种差不多是一样的，而第三种是最好能知道返回值的个数，这样才能一一对应，但是如果个数不一样怎么办呢？

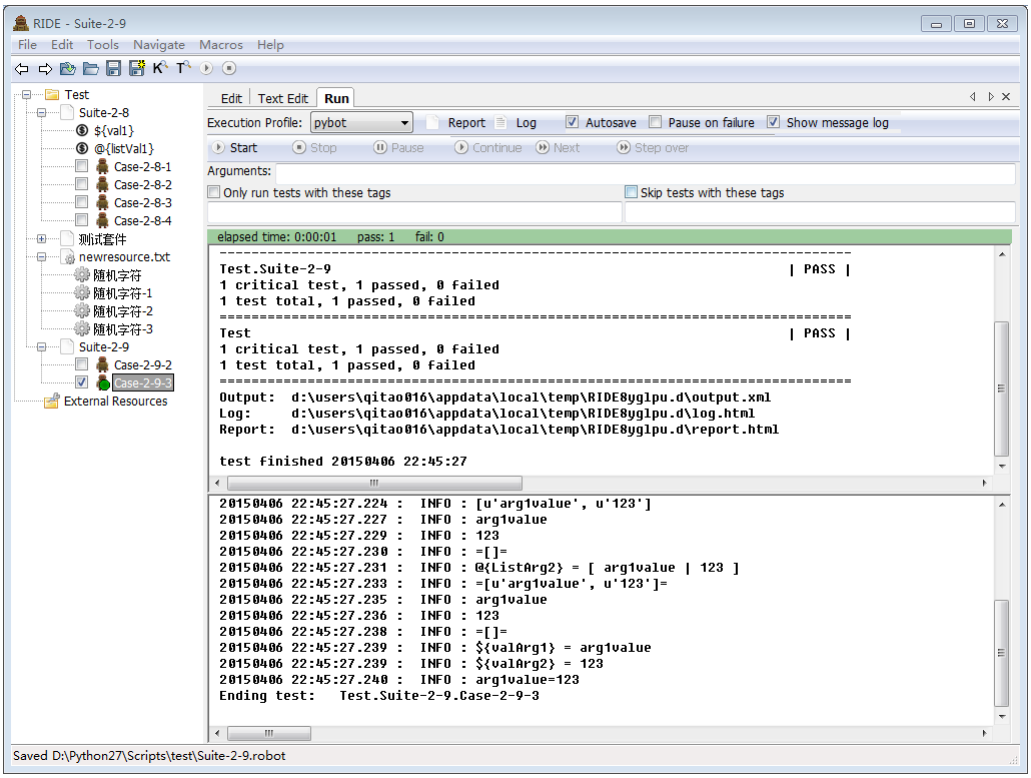

图 2-9-26

第一种情况：返回值个数大于取值变量个数。

实际上在“1）”的方式里已经有了一个类似的，${getArg2}会自动转为 List。这是只有一个取值的变量，那么试试多个的。把“随机字符-2”的 Return Value 加一个，让它返回三个值，如图 2-9-27 所示。

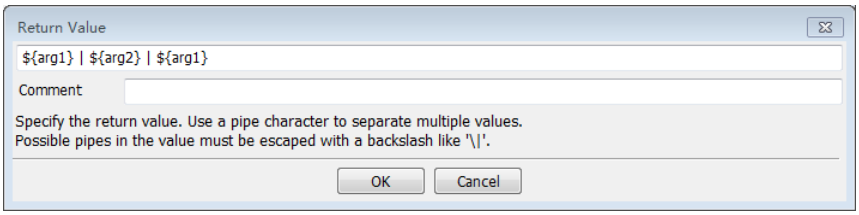

图 2-9-27

在 Case 里还是用 2 个变量取值，脚本不用变，运行一下看看结果，如图 2-9-28 所示。

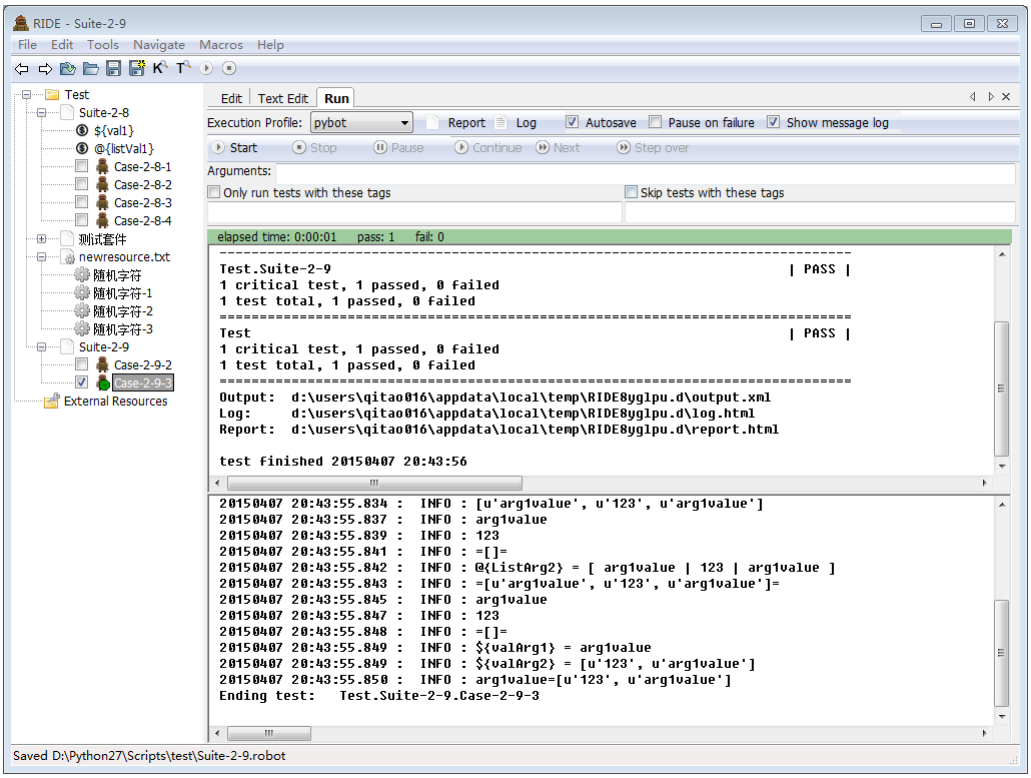

图 2-9-28

它会先把第一个值给了${valArg1}，然后把后面的值给了${valArg2}，于是${valArg2}变成了 List 变量。

所以可以得出结论，对于多个取值变量的个数少于返回值的个数，它会先把前面的值一一对应的给前面的取值变量赋值，这些变量仍然是单值变量，而最后一个变量会转成 List 变量接收剩下的值。有兴趣的读者可以继续增加一下变量个数体验一下。

第二种情况：返回值个数小于取值变量个数。

把“随机字符-2”的 Return Value 改成只有一个${arg1}，如图 2-9-29 所示。

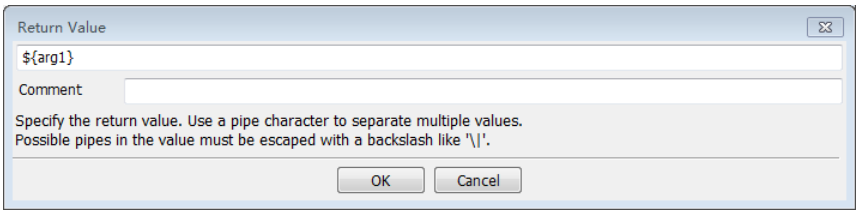

图 2-9-29

然后还是前面的案例保持不变，运行一下看看结果，如图 2-9-30 所示。

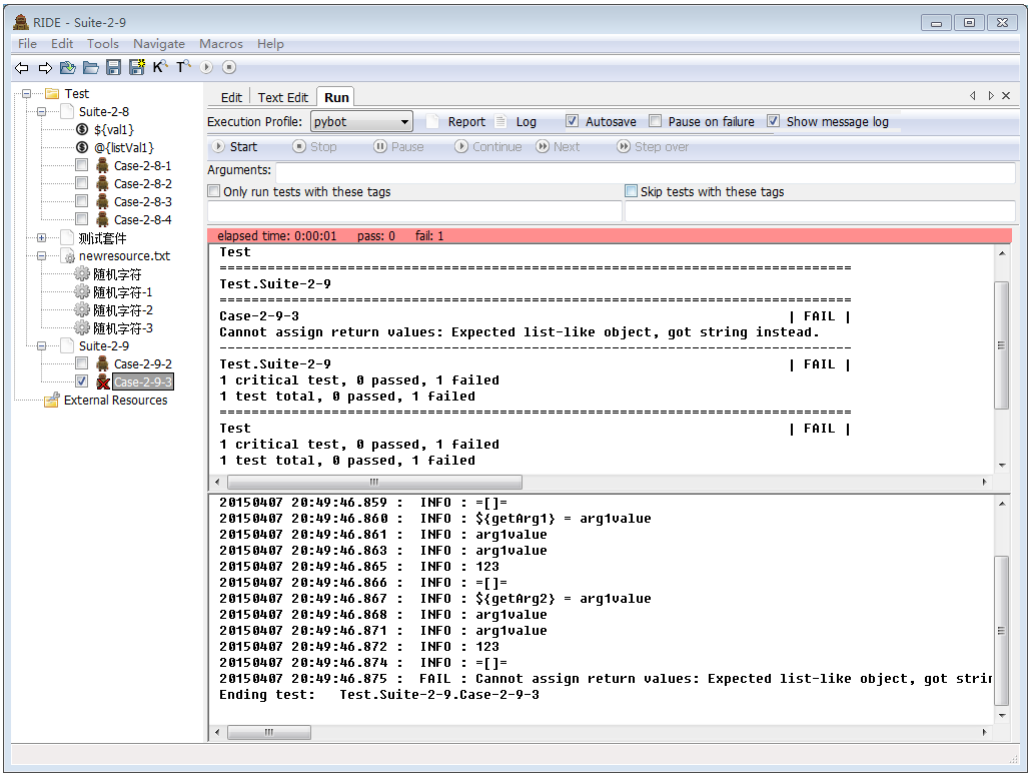

图 2-9-30

这种就会报错了，“Expected list-like object”这里期待的是返回多个值，list-like 的对象，像 List 一样的变量，但是只传回来一个值，两个变量不够分的。

因此，知道返回值的个数比较重要，如果不确定返回值的个数，就可能会出现报错的情况，因此最好使用 List 变量或单个变量来取值，避免出现返回值个数少于取值变量个数

的情况。

演示完成了，笔者还是把“随机字符-2”的 Return Value 改回 2 个，避免给大家的脚本无法执行。

（3）List 变量返回

在“随机字符-3”里把@{arg3}放到 Return Value 里。

因为 List 变量本身就是不确定有多少个成员，所以对于这种返回值，最好使用 List 变量或单个变量来取值（否则可能会出现前面返回值少于取值变量个数的报错）。那么返回值里返回 1 个 List，还是多个 List 都无所谓了，因为它还会组装成一个大的 List。脚本如图 2-9-31 所示。

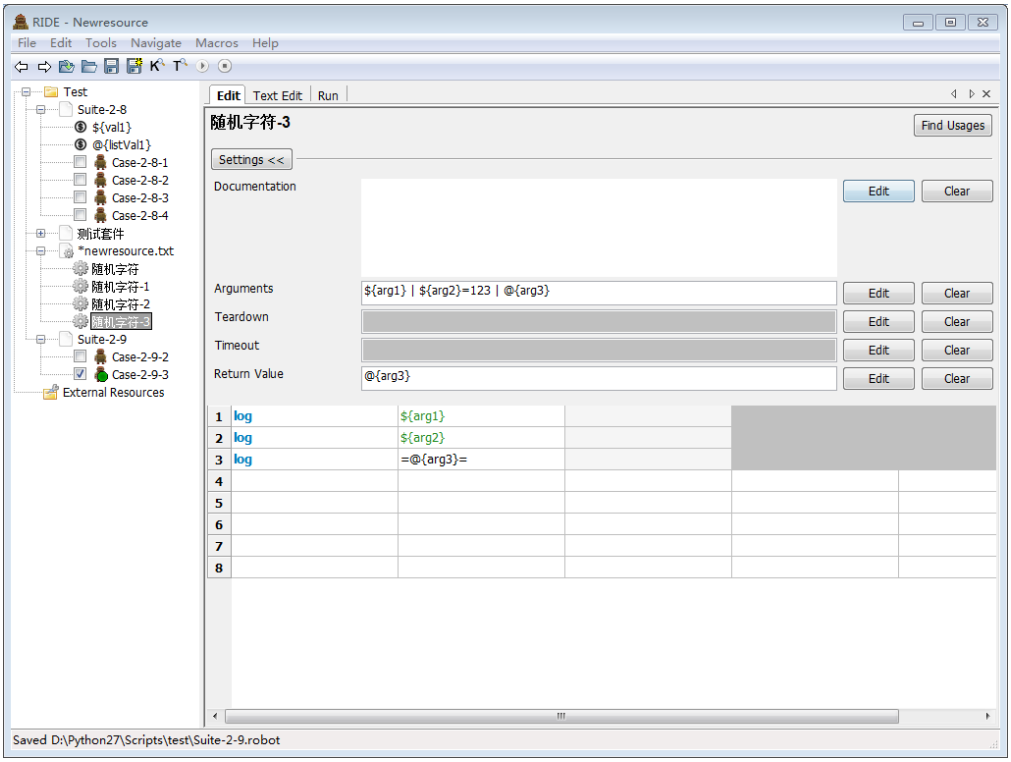

图 2-9-31

在 Case 里同时把两种获取的方法都用上，${getArg3}和@{ListArg3}这两个都用上了，新加脚本见表 2-9-8。

表 2-9-8

${getArg3}	随机字符-3	arg1value		arg3
log	${getArg3}			
@{ListArg3}	随机字符-3	arg1value		arg3
log	=@{ListArg3}=			

最后运行一下，查看结果，如图 2-9-32 所示。

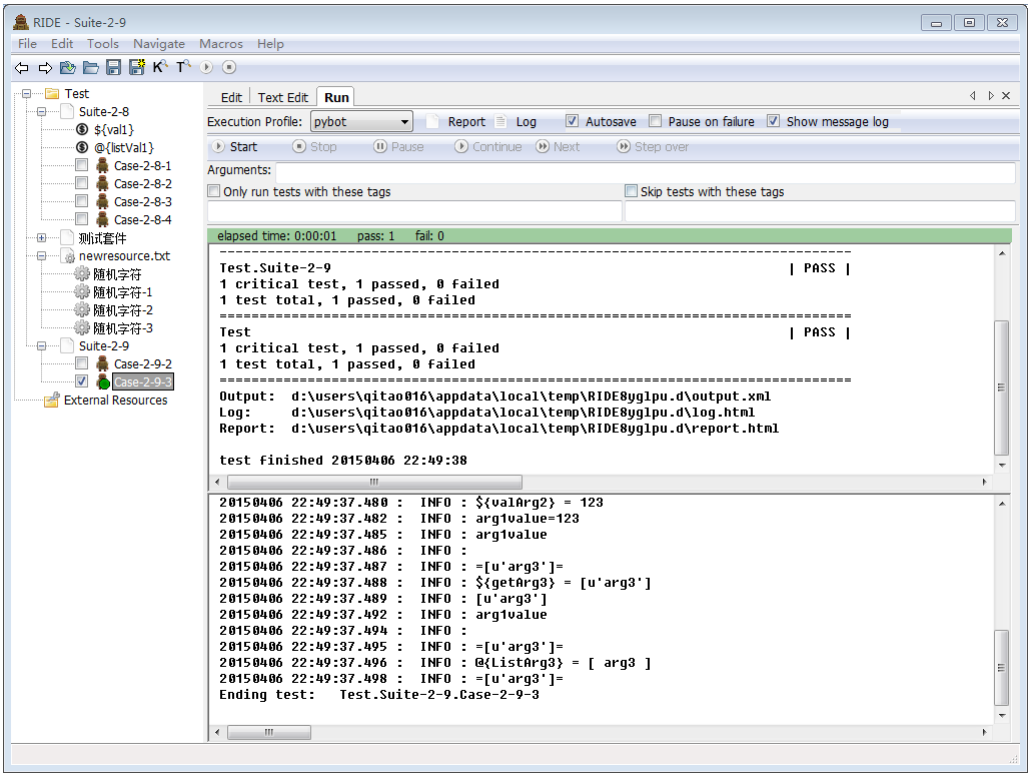

图 2-9-32

正是因为用户关键字有了传入参数和返回值，它就具备了函数的特征，函数能实现的功能，用户关键字一般也能实现。原本想在这里加一节对比用户关键字和系统关键字的，想了一下可能内容过于深入了，笔者会把这节内容放到测试库扩展的地方来给大家演示。

2.10　循环&分支

很多人认为自动化测试里不应该有过多的循环和分支，其实这就和编程的圈复杂度一样，循环和分支多了，不利于让别人阅读你的代码。笔者觉得编程语言里还是要有循环和分支，万一有用呢。当然，只是尽量少用比较好，特别是分支。为什么这么说？因为分支会影响测试结果的判断。比如界面上一个文本框的值是 A，就满足规则 1；如果不是 A 就走规则 2，看起来很符合分支。但是当你其实期待它的值是 A 的时候，根据你的分支走了规则 2，实际应该走规则 1 而没有走规则 1，案例也是没有报错，正常运行，就会把本该是 bug 的地方隐藏过去了。对于自动化案例来说，应该是预期和结果是唯一的，预期这次执行是 A，如果不是 A 就是错误，可能就是 bug。所以对于分支判断来说，如果可能会影响结果的，最好不要加分支判断。

2.10.1　循环

循环其实有个自己的关键字：repeat，但是这个关键字只是单纯的重复执行而已，和我们真正写代码用到的循环还是有很大差异的。

1. 循环 FOR LOOP

这个功能一直就在系统里支持，只是 FOR 的写法有点特殊，要在前面加上冒号。直接按 F5 快捷键，关键字搜索里是找不到的，但是关键字搜索里能找到的是 EXIT FOR LOOP，当时看到这个关键字就想到了应该是有 FOR 循环的，否则也不会有一个退出循环的关键字了。

FOR 循环结构如下：

:FOR 循环变量 IN RANGE end；

:FOR 循环变量 IN RANGE start end [step]。

或

:FOR 循环变量 IN a [b,c,...]。

而循环体为 FOR 下面的缩进脚本，也就是下面第一格空着的。

“[]”里的为非必需参数。

这样写可能大家不一定能看懂，现在直接上例子。

（1）IN RANGE end：

IN RANGE 主要是用一个整数序列做循环的计数器，如果不写 start，只写 end，那就是默认从 0 开始。第二行是循环体，第一格要空出来，脚本见表 2-10-1。

表 2-10-1

:FOR	${i}	IN RANGE	10
	LOG	i=${i}	

IN RANGE 10，运行时就是从 0 到 9，需要注意的就是 RANGE 里不会执行到最后那个数字，也就是 10 是不执行的，到 9 就结束了。如果想执行到 10，那就要写 IN RANGE 11。运行结果如图 2-10-1 所示。

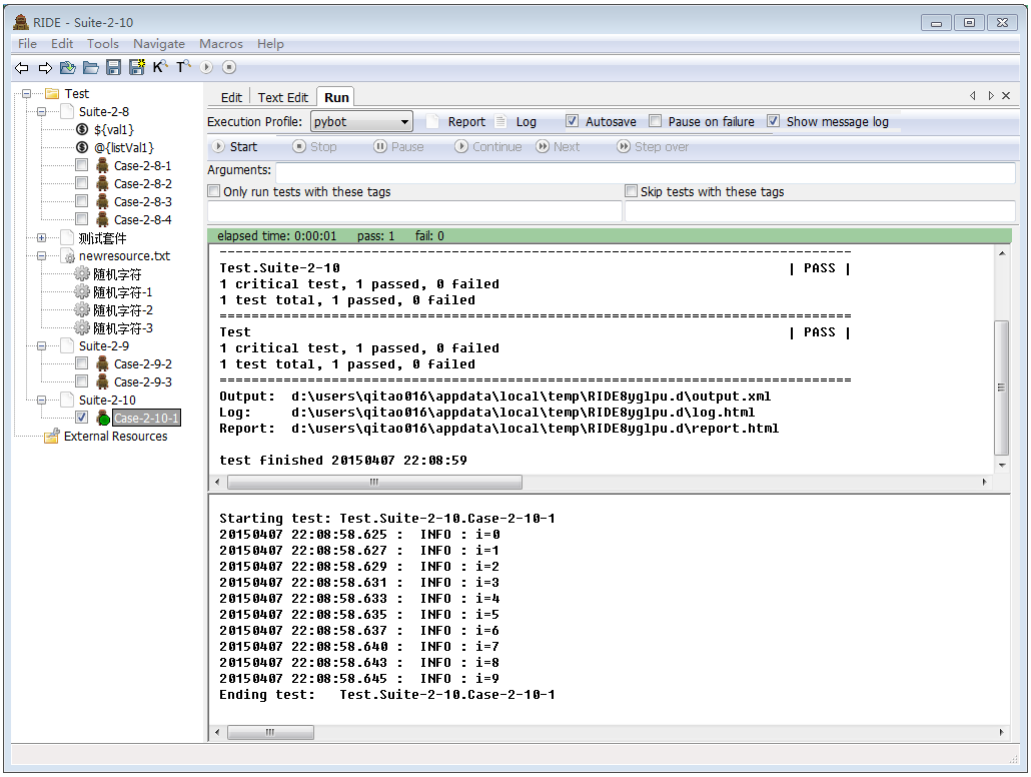

图 2-10-1

总结下来，IN RANGE 后面只写一个数字 N，就是从 0 到 N-1 的整数序列。

（2）IN RANGE start end [step]:

如果不想从 0 开始，想指定起始区间，比如写 IN RANGE 1 10，脚本见表 2-10-2。

表 2-10-2

:FOR	${j}	IN RANGE	1	10
	LOG	j=${j}		

运行结果如图 2-10-2 所示。

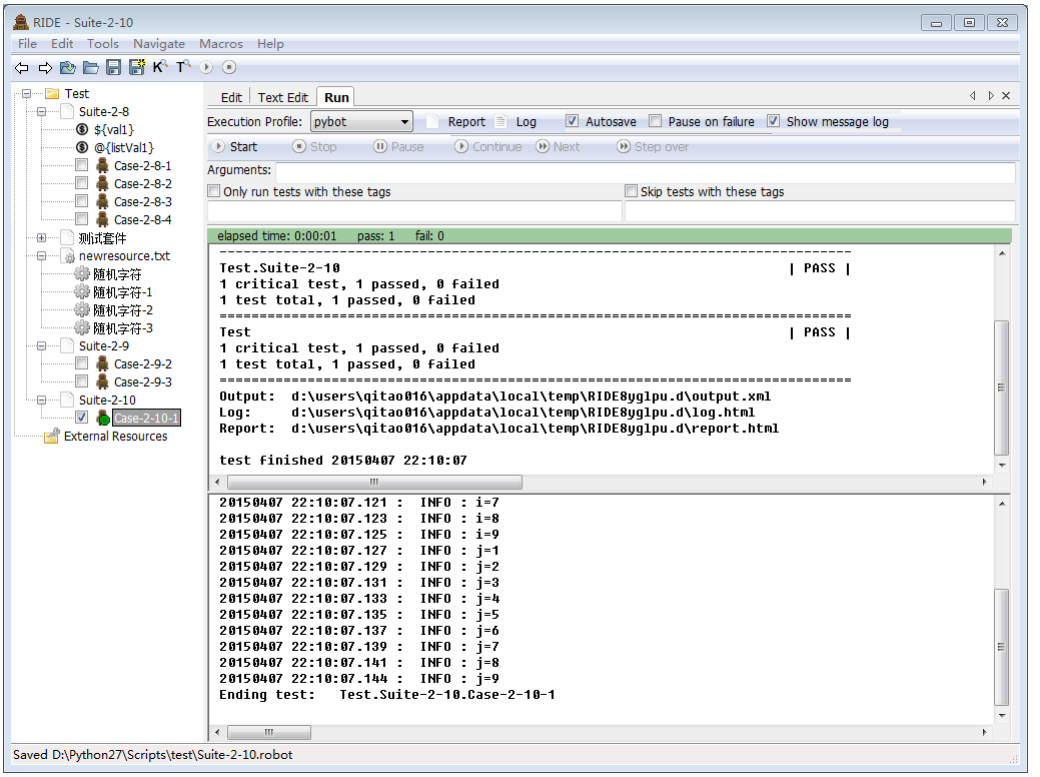

图 2-10-2

当然，既然是序列也可以加 step 步进，即序列的间隔。step 默认是+1 的，如果要加 step，就必须把 start 和 end 的数字都写上，然后在后面写上 step。可以是正数，也可以是负数，脚本见表 2-10-3。

表 2-10-3

:FOR	${k}	IN RANGE	10	1	-2
	LOG	k=${k}			

如图 2-10-3 所示，step 是-2，看一下执行结果。

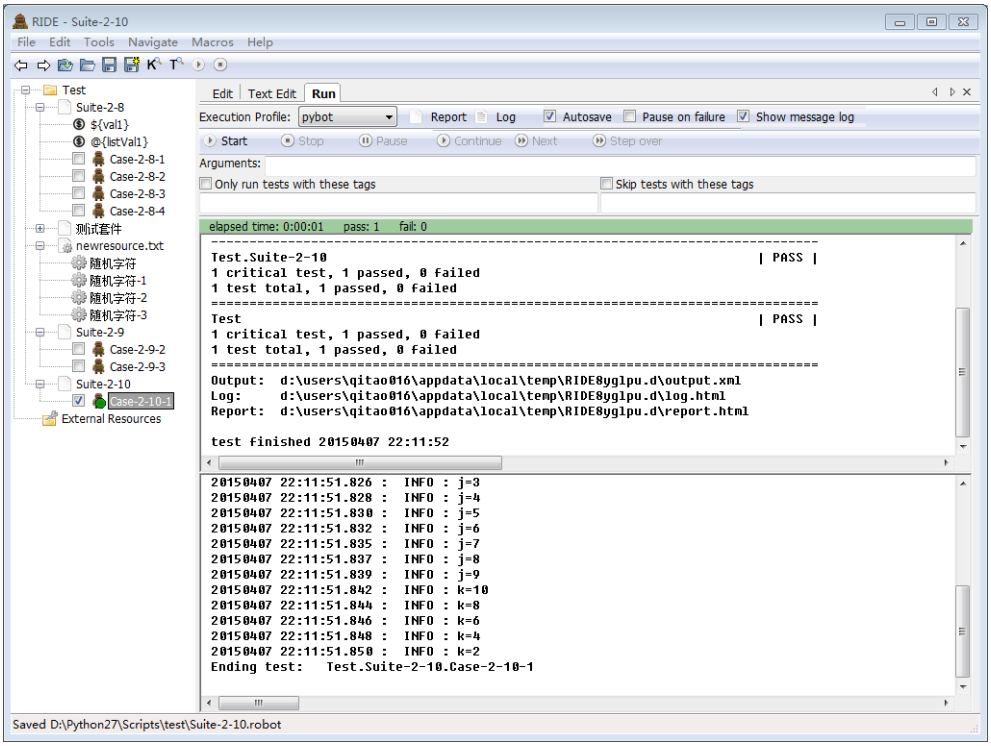

图 2-10-3

如果你的 start 大于 end，而 step 是正数，或者 start 小于 end，而 step 是负数，这样的循环都是无法执行的。

（3）IN:

IN 的用法就比较自由了，可以把 IN 后面的内容理解为一个 List 变量的全部元素，这个循环相当于遍历每个元素。这里的元素可以是字符串，也可以是数值。

所以笔者下面放了 2 个例子，可以像笔者那样把元素直接写出来，也可以直接放一个 List 变量，这两种形式的脚本都做了，见表 2-10-4。

表 2-10-4

:FOR	${m}	IN	A	B	C
	LOG	m=${m}			
@{listVal}	Create List	1	2	F	
:FOR	${n}	IN	@{listVal}		
	LOG	n=${n}			

${m}和${n}的两个循环就是两种不同用法，运行结果如图 2-10-4 所示。

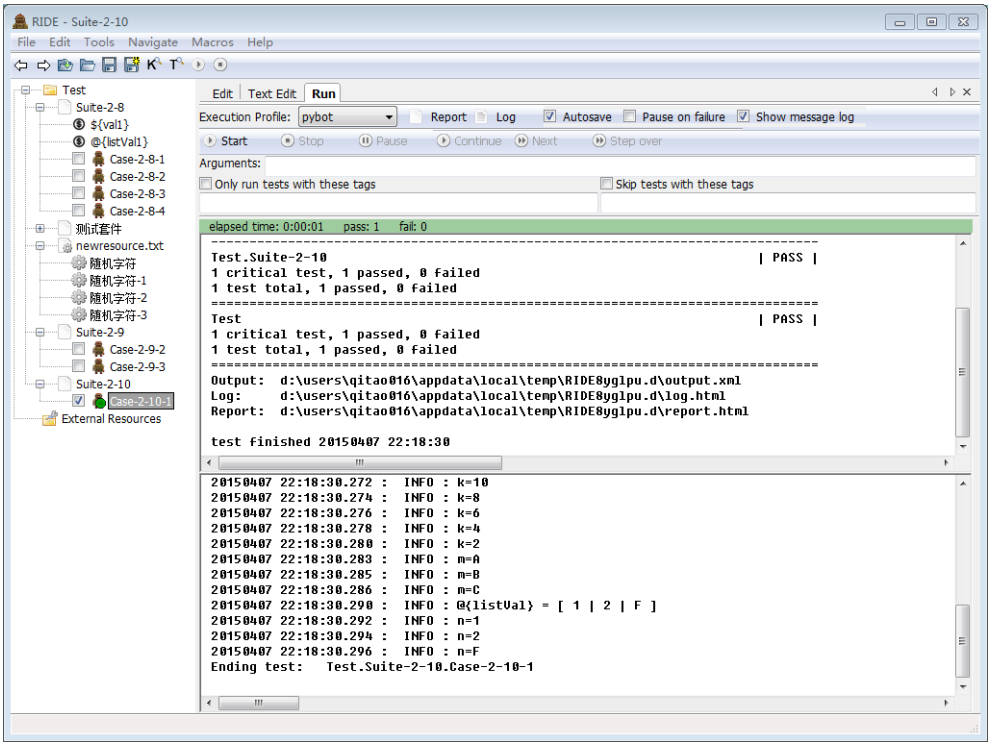

图 2-10-4

具体使用 IN 还是 IN RANGE，看具体需要，这里只提供了例子，大家自行选择具体方法。比如你写“IN 1 2 3 4 5 6 7 8 9 ”就不如直接用“IN RANGE 1 10”，效果都一样。

2. 双重循环

以前写 vbs 的 FOR 循环可以这样写：

```
FOR i=1 to 10
   FOR J=1 to 8
      do something
   next J
next i
```

但是目前 RF 这里的 FOR 循环无法直接支持这样做，只能间接地做。

首先要新增一个 User Keyword，为了演示方便，笔者直接在 Suite 下新增了一个 ForJ 的关键字，加上一个传入参数，加上个小循环，循环体里把${arg1}和${j}的值都打印出来，脚本见表 2-10-5。

表 2-10-5

:FOR	${j}	IN RANGE	2
	LOG	arg=${arg1};j=${j}	

整个关键字包括 Arguments 的截图，如图 2-10-5 所示。

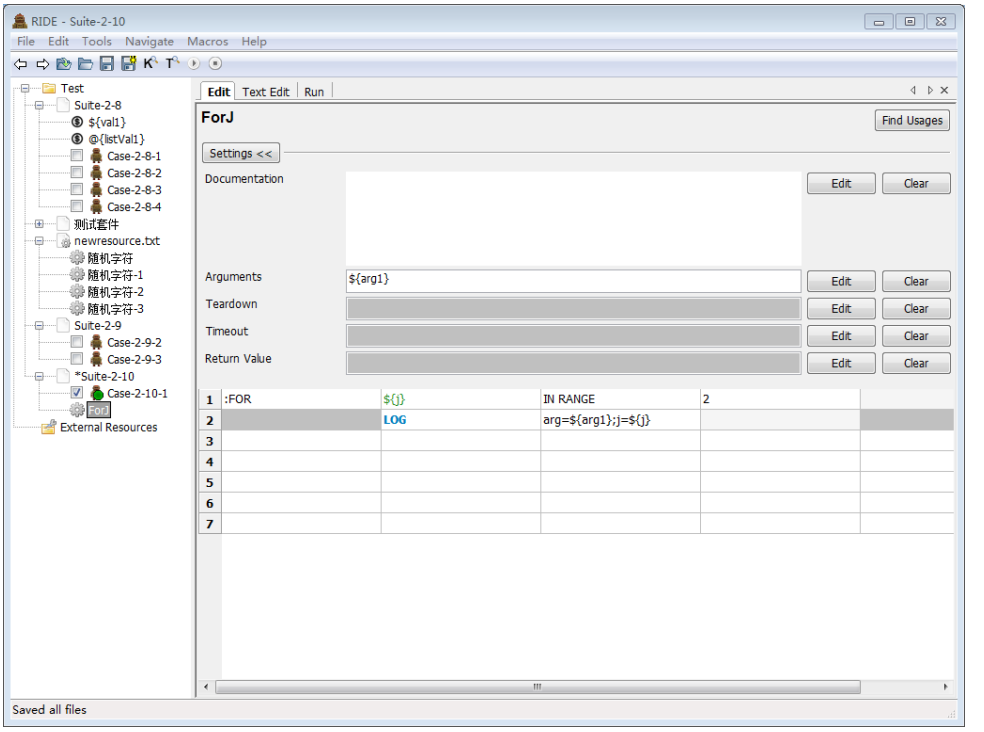

图 2-10-5

然后在 Case 里新增一个循环，在循环体里做两件事情，一个是打印${i}，另一个是调用 ForJ，并把${i}传给它，脚本见表 2-10-6。

表 2-10-6

:FOR	${i}	IN RANGE	3
	LOG	i=${i}	
	ForJ	${i}	

运行一下看看结果，如图 2-10-6 所示。

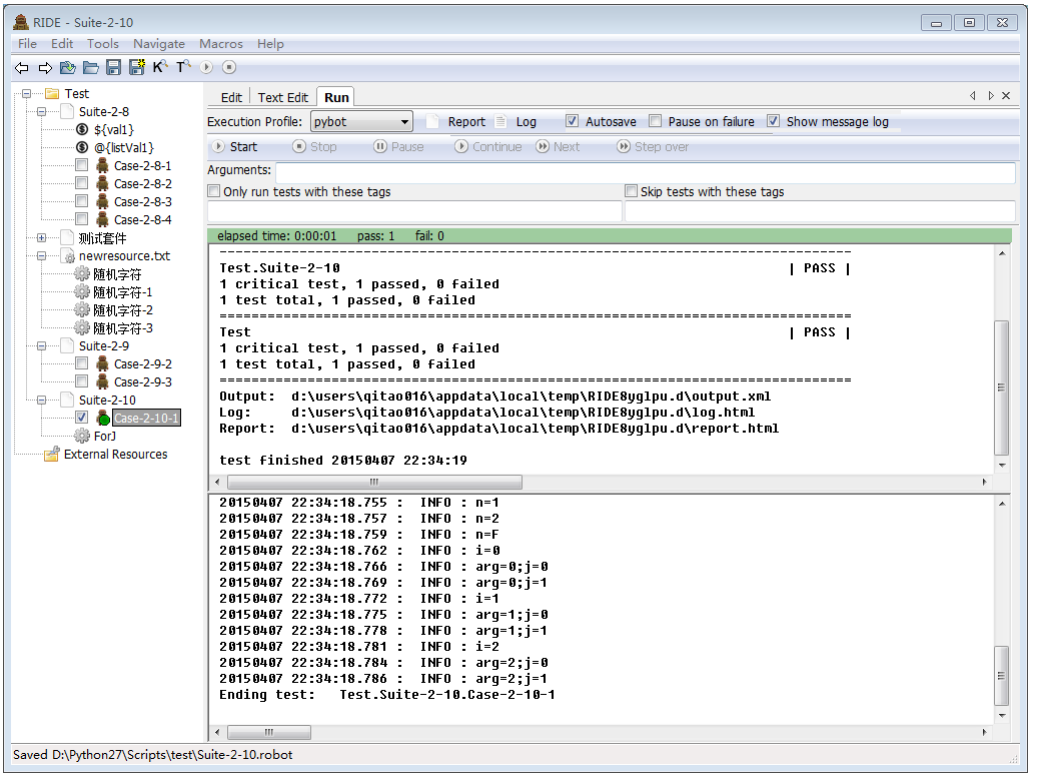

图 2-10-6

这样就实现出了上面的那种效果了。

需要注意的是最后一个例子的循环体是有两行的，如果不小心忘记了第二行前面要空格了，那么这一行就不在循环体内了，比如脚本是这样的，见表 2-10-7。

表 2-10-7

:FOR	${x}	IN RANGE	3
	LOG	x=${x}	
ForJ	${x}		

运行后的结果，如图 2-10-7 所示。

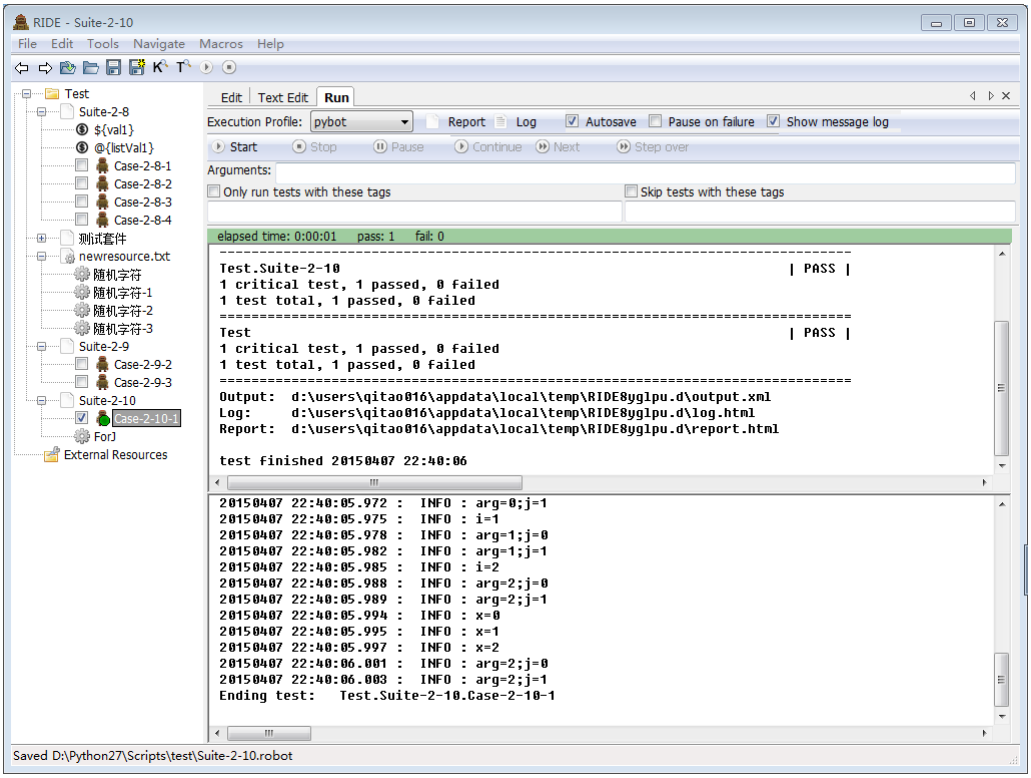

图 2-10-7

可以看到，ForJ 是在循环体外执行的，因为它只执行了一次，使用的${x}的值也是最后一次的值。这一点需要大家注意。

2.10.2 分支

分支其实是主要通过关键字完成的，即 Run Keyword If。

在 Robot Framework 2.7.4 之前的版本，我们要想写 IF 比较容易，可以直接使用 Run

Keyword If 就行了，但是如果想写 ELSE IF 和 ELSE，就不是那么方便了，以前的版本想写判断分支，就必须要再写一个 Run Keyword If，然后写不同的条件。

好在从 2.7.4 的版本开始新增了这个判断分支结构的扩展，也就是在 Run Keyword If 的语句基础上，集成了 ELSE IF 和 ELSE。

1. 标准分支写法

分支的标准结构如下：

Run Keyword If　条件 1　　do action1

...　ELSE IF　条件 2　　do action2

<...　ELSE IF　条件 X　　do action X >

...　ELSE　do action N

先写个例子好了，这也是参考更新文档里的例子。脚本见表 2-10-8。

表 2-10-8

${a}	Set Variable	0		
${b}	Set Variable	5		
#多行写法				
Run Keyword If	${a} >= 1	log	1	
...	ELSE IF	${b}<=4	log	2
...	ELSE	log	3	

其中 ELSE IF 和 ELSE 前面的“...”这三个点是必须加的，否则 RF 无法识别。这里的意思就是先判断如果${a}的值大于等于 1，log 就打印 1，否则如果${b}的值小于等于 4，就打印 2，否则就打印 3。

实际的案例，如图 2-10-8 所示。

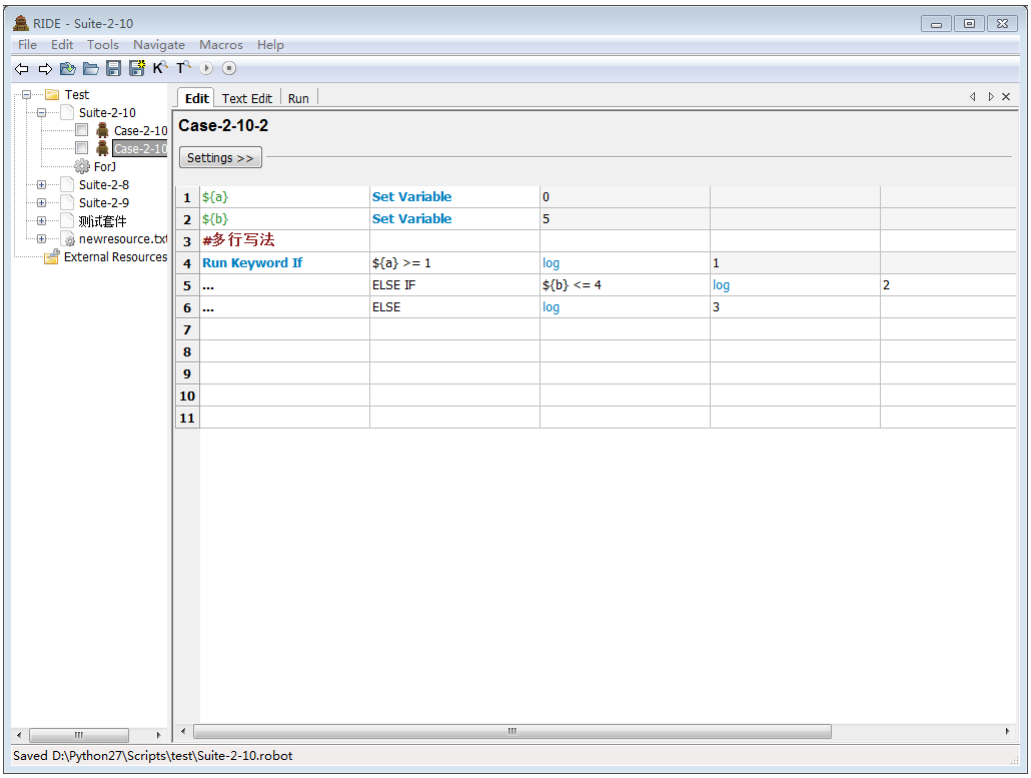

图 2-10-8

然后运行一下，运行结果如图 2-10-9 所示。

大家可以自己练习修改${a}和${b}的变量值，来验证一下这个分支判断是否正确，结果笔者在这里就不展示了。

上面给的结构只是一个标准的结构，你可以根据自己的需要进行改造。

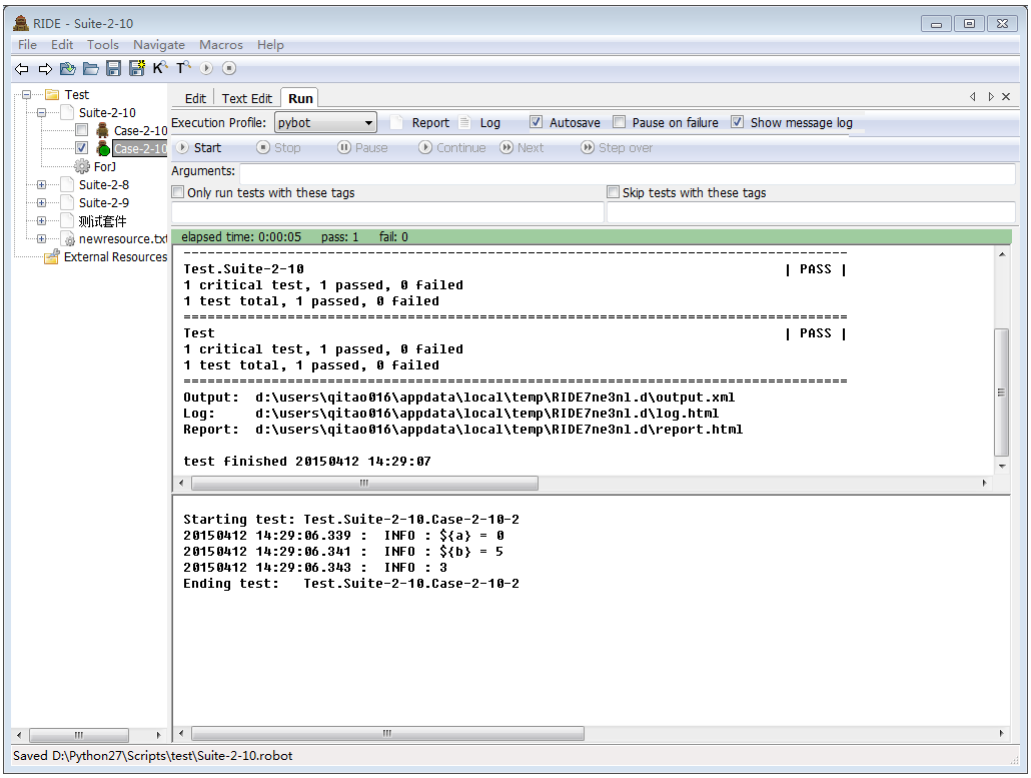

图 2-10-9

比如中间的 ELSE IF 是可以有多个的，写多一些就有点像 CASE 分支了，见表 2-10-9。

表 2-10-9

#多行写法				
Run Keyword If	${a} >= 1	log	1	
...	ELSE IF	${b}<=4	log	2
...	ELSE IF	${b}<=7	log	5
...	ELSE	log	3	

这里多个 ELSE IF 的笔者给出写法，例子中就没有加了，大家可以自己练习的时候加上。你可以只写个 ELSE IF，或者只写 ELSE，看你的需要。

如果只有 ELSE 的话，也可以写成单行模式，见表 2-10-10。

表 2-10-10

Run Keyword If	${a} >= 1	No Operaton	ELSE	log	6

实际的案例，如图 2-10-10 所示。

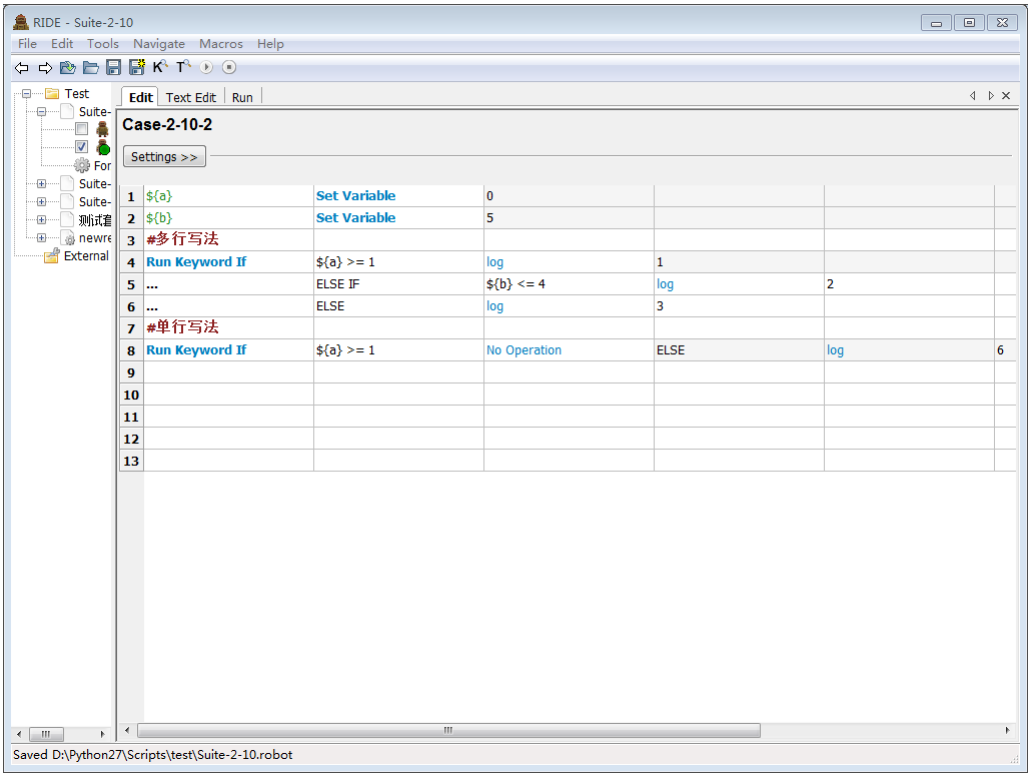

图 2-10-10

运行结果如图 2-10-11 所示。

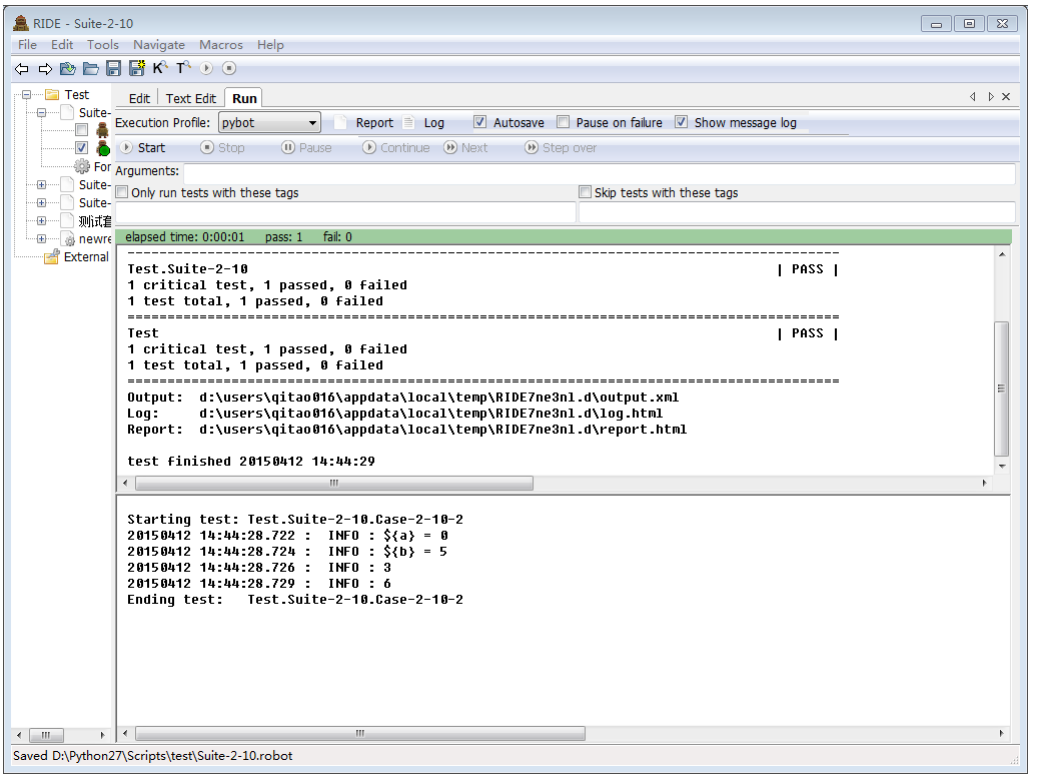

图 2-10-11

虽然看起来写了两种写法，多行模式和单行模式，但是实际上它们都是一样的，因为在 RF 里每一行是一个脚本，虽然你写成了多行，但是实际上还是单行。你不信？重新在 RIDE 里加载一下刚才的案例，如图 2-10-12 所示。

重新打开后，多行的也变成单行了，这其实是 RIDE 自己处理的。

总体来说，这里的关键字就是判断当条件满足的时候，执行一个关键字，因为 Run Keyword If 本身也是关键字，所以还有一种比较复杂的写法，是在 ELSE IF 或者 ELSE 里继续加上 Run Keyword If 来构造一个复杂的判断，但笔者觉得大可不必，完全可以把判断条件整合一下，尽量写简单一点，遮罩复杂逻辑不太适合在书中展现，如果有兴趣的读者可以去笔者的博客上看一下。

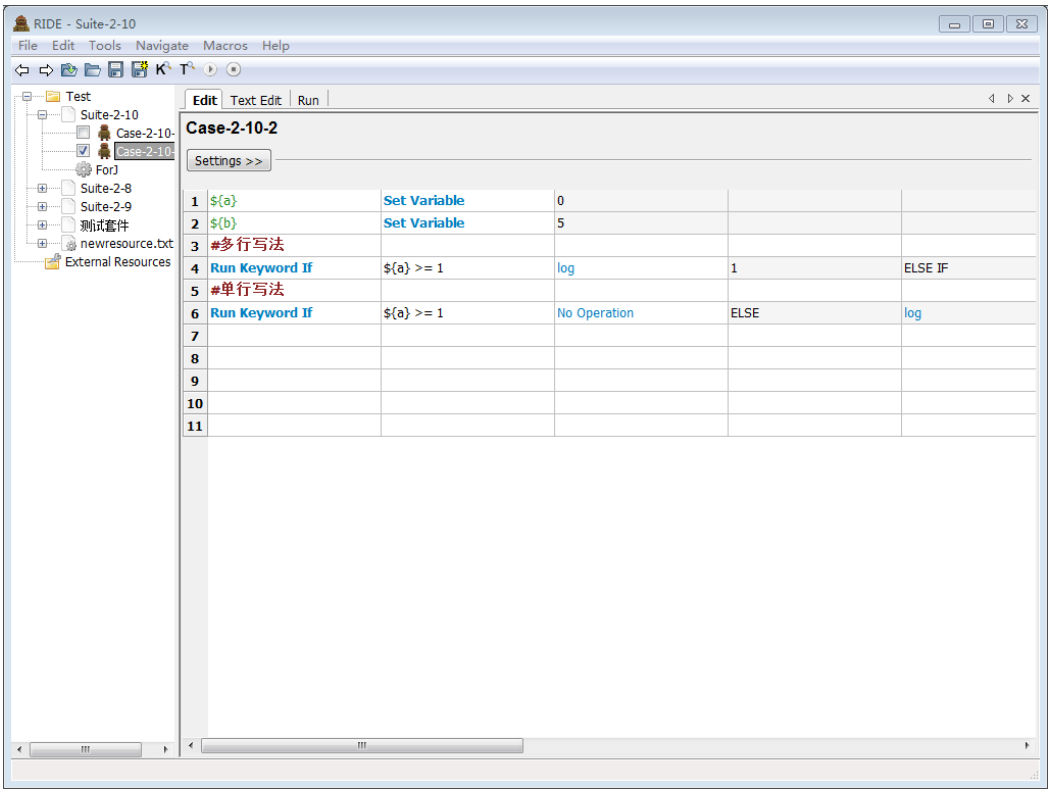

图 2-10-12

2. condition 条件

基本上分支说的差不多了，笔者还想补充说一下 condition 条件的写法，条件主要就是判断当前表达式的值是 True 还是 False。

可以先按 F5 快捷键查看一下 Run Keyword If 的帮助说明，注意其中这 2 段：

> Runs the given keyword with the given arguments, if `condition` is true.
>
> The given `condition` is evaluated similarly as with `Should Be True` keyword, and `name` and `*args` have same semantics as with `Run Keyword`.

这里的意思是说，如果条件为真的时候，运行给定的关键字和参数。

而 condition 会被计算求值，类似于 Should Be True 这个关键字。就看到这里就行了。实际上 condition 会用 Python 语法进行判断，相当于执行了 evaluate（以后会讲到这个“神

奇”的关键字）。

如何判断 condition 的值是真是假？

（1）单个对象

如果表达式里只有单个对象，那就要看这个对象的值了。如果值是 None、False、0、空（'',[],(),{}）等，这些值就是 False 的，除此以外的值一般都是 True。

（2）布尔操作符

常用的就是 and/or/not，含义如下：

```
x or y   if x is false, then y, else x
x and y  if x is false, then x, else y
not x    if x is false, then True, else False
```

表达式里如果有布尔操作符，那么先要看表达式最终的值是哪个，然后在看值的真假，决定了 condition 的真假。

（3）比较操作

最常用的应该就是比较操作了，像前面的例子里也是用的比较操作，比较操作常见的有如下写法：

```
<    strictly less than
<=   less than or equal
>    strictly greater than
>=   greater than or equal
==   equal
!=   not equal
is   object identity
is not   negated object identity
```

其中“!=”也可用“<>”，但是建议都用“!=”。

如果是 List 变量，也可以用 in 或 not in 来写表达式。另外在表达式里加上数学运算也可以，这里就不一一列举了。

结合以上这些操作符，应该足够满足大家在一般情况下的使用了，如果想完整了解这一块的内容，可以查看一下官方文档：

https://docs.python.org/2/library/stdtypes.html#truth

2.10.3 二者结合

循环和分支也可以结合起来用，比如下面这个案例，脚本见表 2-10-11。

表 2-10-11

:FOR	${i}	IN RANGE	10
	LOG	i=${i}	
	Run Keyword If	${i}>=7	Exit For Loop

我们用了 Run Keyword If 判断${i}大于等于 7 的时候退出循环，调用 Exit For Loop，其实这里还有个简单的用法，就是 Exit For Loop If，案例脚本见表 2-10-12。

表 2-10-12

:FOR	${i}	IN RANGE	10
	LOG	i=${i}	
	Exit For Loop If	${i}>=7	

执行一下案例看看结果，如图 2-10-13 所示。

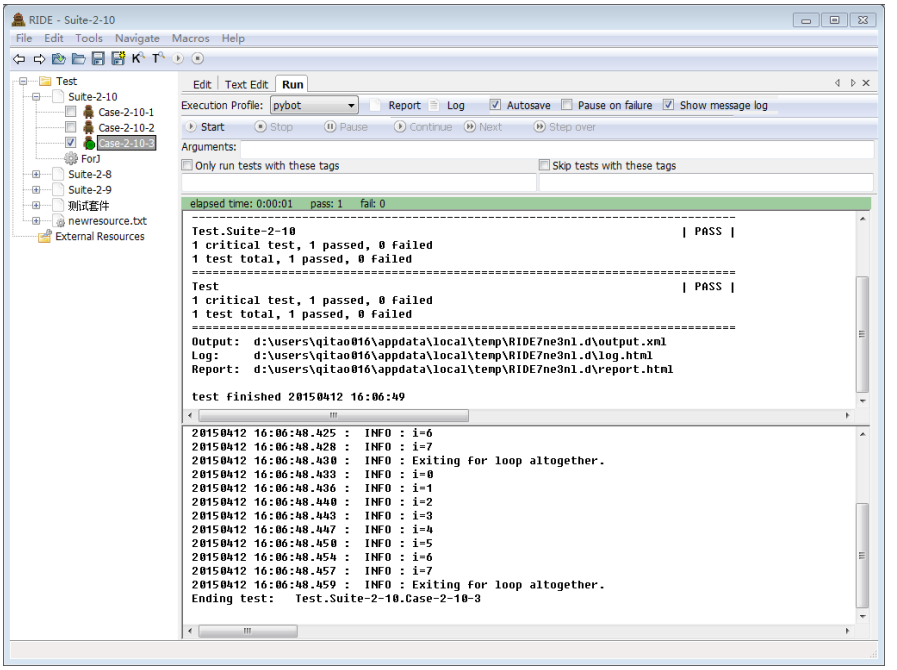

图 2-10-13

最后还是要再次强调一下本章节开头说过的话题，只建议在不影响测试结果的情况下，或者在万不得已的情况下才加判断，否则就可能会发生一些不好的事情。

2.11　运行界面

Run 运行界面，其实在前面的例子中都看过不少，这个界面里其实还有很多操作没有细说，在这里一起过一遍。

就用前面一节的这个图（图 2-10-13）来看吧，如图 2-11-1 所示。

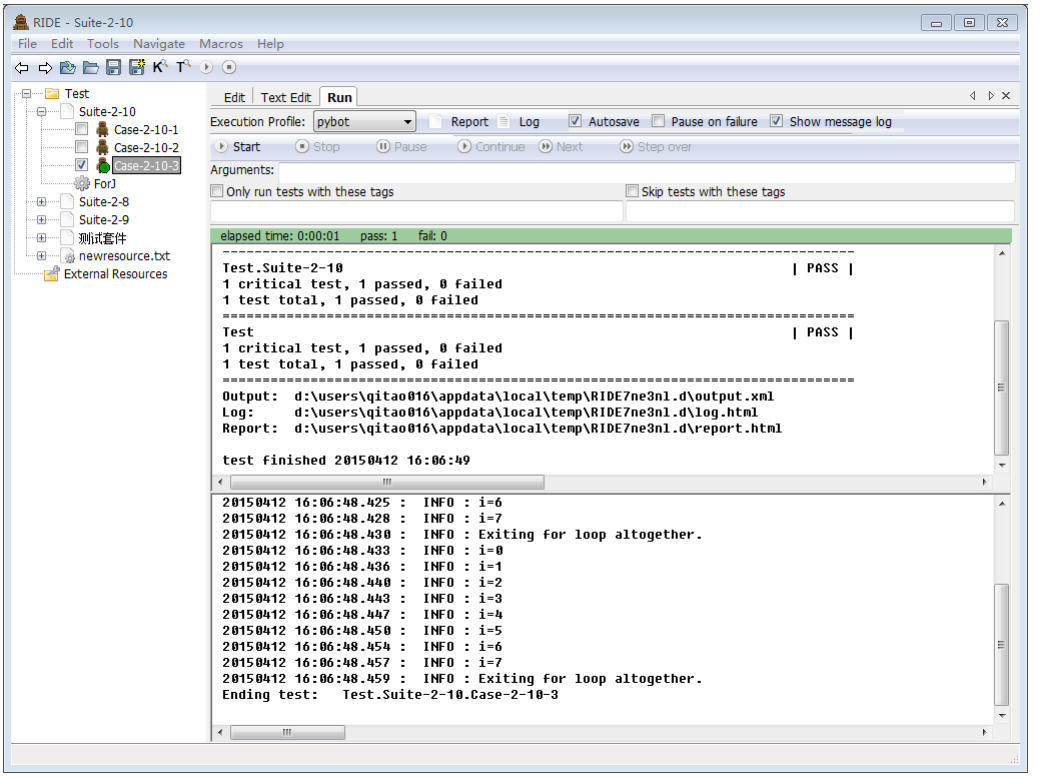

图 2-11-1

在 Run 的界面里，第 1 行是一些配置选择项。

- Execution Profile：选择使用哪个工具来执行案例，默认是 pybot。此外还有 jybot 和 custom script，jybot 需要安装 jython 支持；custom script 就是用自定义的脚本来

执行，对于初学者来说不用动它，默认 pybot 就行了；

- Report & Log：这两个按钮就是查看执行结果的报告的，在前面的章节中已经演示过了；
- Autosave：自动保存。如果不勾选的话，每次执行案例时，如果脚本没有保存都会提示是否保存。比较懒一点的朋友就勾上吧；
- Pause on failure：失败时暂停。调试时可以用一下，不过笔者觉得略微不太方便；
- Show message log：显示日志信息打印。就是最底下的区域，如果不勾选的话，就看不到运行期间打印的日志了，默认勾选就行了。

第 2 行就是调试用到的一些操作了，Start、Stop、Pause、Continue 应该不用说了吧，后面 2 个 Next 和 Step Over 是在暂停情况下，用来单步执行调试的。

第 3 行就是 Arguments 了，前面给变量赋值的时候，有一个就是运行时赋值的例子，当时就是在这里增加了“-v”的参数，这里可以加很多的参数，这里就不全部列举了，大家可以在命令行里执行 pybot.bat --help 来查看都有哪些参数（Mac 环境下执行 pybot --help）。

第 4 行就是在讲测试案例时提过的 Tag 的使用，可以给案例加上不同的 Tag 来进行标记，然后在这里或者 pybot 的参数来控制跳过哪些案例、只执行哪些案例。

- Only run tests with these tags：只执行有这些 tags 的测试案例；
- Skip tests with these tags：跳过有这些 tags 的测试案例。

来用一个例子演示一下，新增一个 Suite-2-11，在下面新增 4 个 Case，每个 Case 里只有一个 log 打印的脚本。给这 4 个案例前面的复选框打勾，然后分别给下面 4 个 Case 加上不同的 Tags:

- Case-2-11-1：RunA RunAll；
- Case-2-11-2：RunB RunAll；
- Case-2-11-3：RunA RunB RunAll；
- Case-2-11-4：RunAll。

标签的含义可以假设为 RunA 是进行 A 轮测试要执行的案例；RunB 是进行 B 轮测试要执行的案例；RunAll 是全回归要执行的案例，如图 2-11-2 所示。

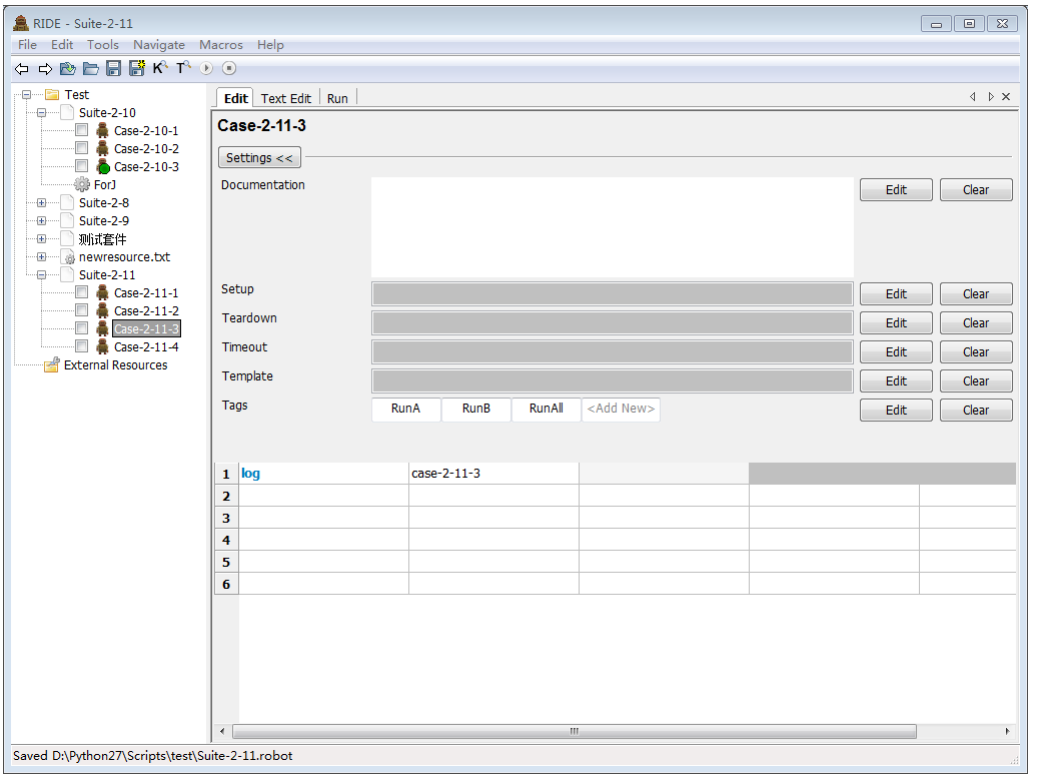

图 2-11-2

在 Run 界面，在“Only run tests with these tags（后面简写为 Only run）”文本框中写 RunA，在“Skip tests with these tags（后面简写为 Skip）”文本框中写 RunB。如果只勾选了“Only run”复选框，没有勾选“Skip”复选框，那么此时运行案例就会在这 4 个案例中找出有 RunA 标记的案例，也就是 1 和 3 会被执行；如果只勾选了“Skip”复选框，没有勾选“Only run”复选框，那就会在这 4 个案例中有 RunB 标记的案例，也就是 1 和 4 会被执行。那么如果两个都勾选会是什么样呢？那先看有 RunA，只有 1 和 3，然后剔除 RunB，就把 3 剔除了，所以这里的结果是只有 1 会被执行，如图 2-11-3 所示。

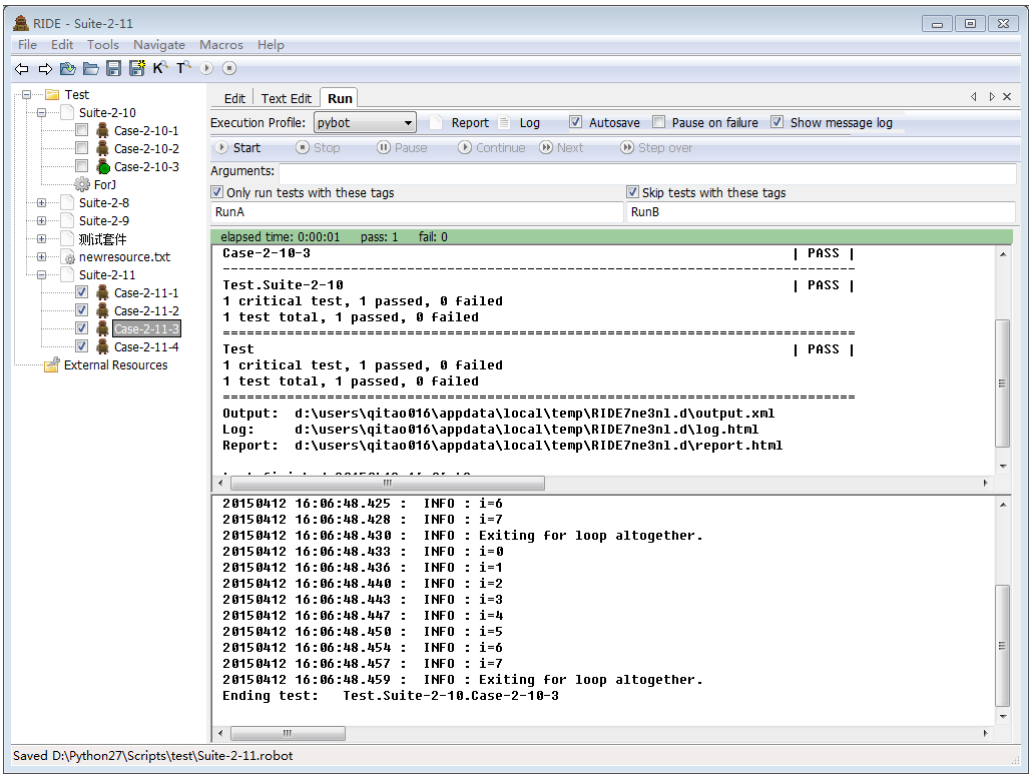

图 2-11-3

两个都勾选来执行一下，如图 2-11-4 所示。

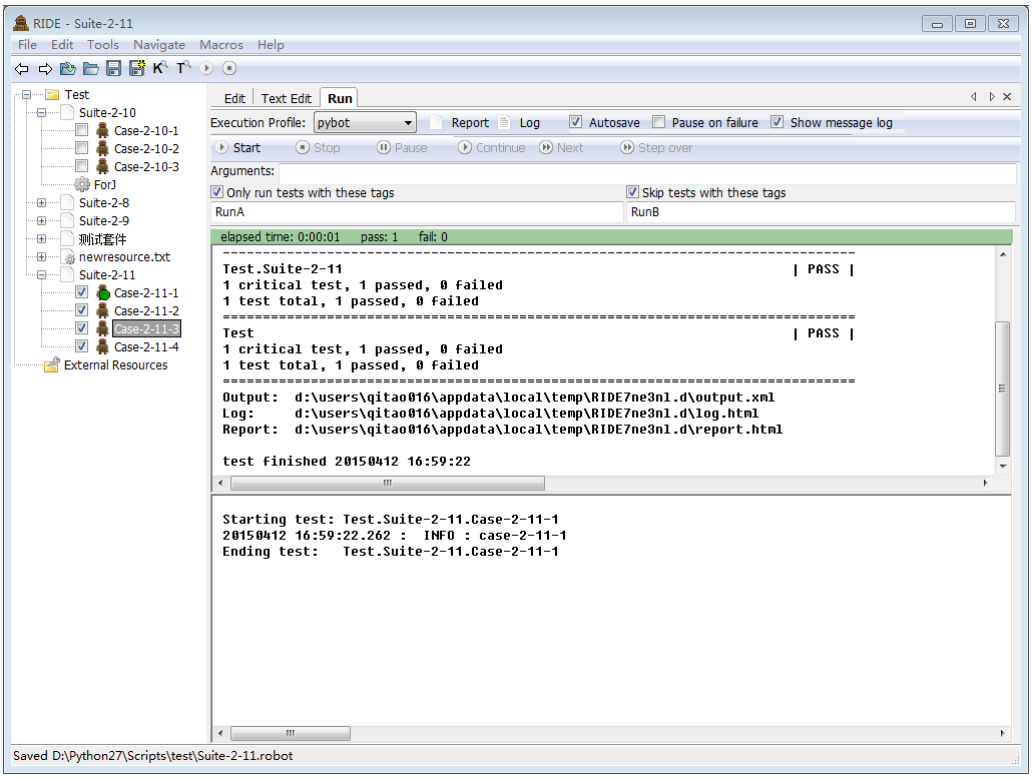

图 2-11-4

这里可以看出，如果一个案例同时有“Only run”和“Skip”的 Tags，比如第 3 个案例，它有 RunA 和 RunB，它还是优先会被“Skip”的剔除掉。所以如果两个同时都选了，“Skip”的优先级高于“Only run”。

这里把“Skip”的 Tags 换一下，换成 RunAll 会是什么情况呢？如图 2-11-5 所示。

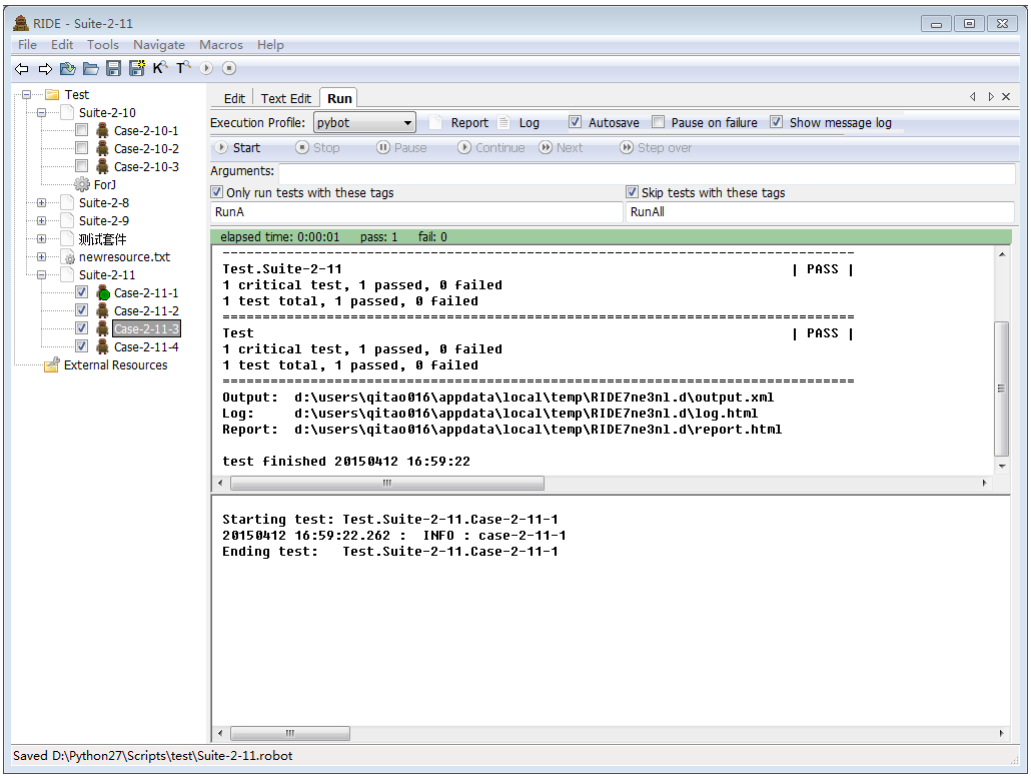

图 2-11-5

运行一下看看结果，如图 2-11-6 所示。这里系统就提示有错误了，这个套件下不存在有 RunA，没有 RunAll 标记的测试案例。

这里主要演示了一下 Tags 的使用，合理利用 Tags 来规划自动化测试案例，可以更方便地执行所需的案例。

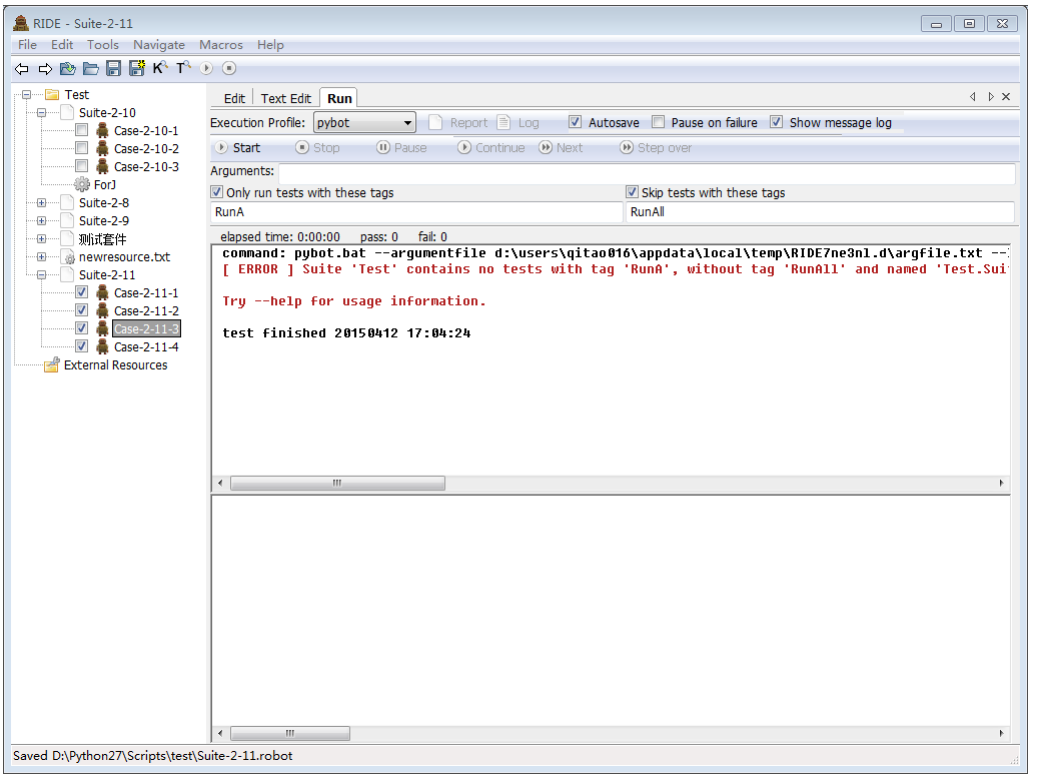

图 2-11-6

2.12　小结

前面介绍的这些内容都是基于 RF 2.8 版本的，在写书期间，RF 发布了 2.9 alpha1 版本，对大家其实是个好事，但是对笔者写书来说是个很郁闷的事情，因为有部分前面写好的内容要进行修改。思考再三，笔者决定还是先放到这个章节里来说明一下更新点。如果本书出版后正式版还没有发布的话，建议大家不要急着更新 2.9 版本，特别是用 RIDE 写案例的读者，因为至少 alpha1 版本 RIDE 是不支持其中新增的修改的部分，具体修改点笔者在后面会介绍，所以有可能你写了正确的语法，RIDE 不识别，不过执行的时候 RF 会识别的。

第一个更新点，最大的亮点就是 Dictionary 变量出现了，在前面变量的章节里，其实已经提到了 Dictionary 变量，但是当时没有细讲。Dictionary 变量之前就有，只不过是用

${Dict}这样的 Scalar 变量形式来保存的。在 RF 2.9 alpha1 的版本里，Dictionary 有了自己的变量标识符“&”，以后就可以用&{Dict}来作为 Dictionary 变量使用了。

第二个更新点，紧跟着 Dictionary 变量，“**kwargs”这种 Python 的参数也可以用在用户关键字上了，以&{Dict}变量的形式。之前看到用户关键字的参数，如果前面没有“*”就是 Scalar 变量参数，有“*”就是 List 变量参数，而还有一种 Python 支持的参数是“**kwargs”，这个语法之前在 Python 里可以写，但是在用户关键字上没法支持。现在有了&{Dict}变量了，就可以一样的在用户关键字的参数里加上这种“**kwargs”变量了，主要是用&{Dict}变量来作为参数。后面具体讲到 Dictionary 时会详细介绍一下。

第三个更新点，其实初学者可以先不用了解，主要是关键字的名字里加上了一些嵌入式参数，可以让关键字的名字根据参数动态变化。说起来也是比较绕口，等 2.9 正式版推出后我们再来研究这一块内容吧。

第四个更新点，Scalar 变量和 List 变量将会使用同一个 namespace 命名空间存储变量的值。前面 3 个更新点，其实都是笔者还没有写到的地方。而这一个更新点，其实笔者前面已经写完了（这也是笔者纠结要不要修改前面内容的地方）。在讲到 Scalar 变量和 List 变量转换时，假设你用两个一样变量名的不同变量，比如${arg1}和@{arg1}，如果你分别对它们进行了不同的赋值，此时它们的值会不一样，这其实是因为它们之前没有保存在同一个命名空间里，而这个更新点会对前面的结论有很大影响，它们使用同一个命名空间来保存变量的值，如果还是前面的例子的话，那么如果你进行了不同的赋值，此时无论是${arg1}还是@{arg1}，都只会有最后一次赋值的内容，也就是它们会一直保持统一，随时可以互换。从某种角度来说，笔者认为是个好事，因为前面笔者也说过，让大家注意两个变量值不同这一点，现在笔者觉得可以不用特意注意了，因为它们保持统一了。

除了上述比较大的更新点以外，还有很多更新点，这里就不一一列举了，大家可以查看更新说明，地址如下:

https://github.com/robotframework/robotframework/releases/tag/2.9a1

有可能大家看到本书的时候，已经不是这个 alpha 的版本了，可能有更新的出来了，请到 release 下查看最新的。

第二部分　小乘篇

- 第 3 章：Web 自动化测试
- 第 4 章：C/S 自动化测试
- 第 5 章：数据库自动化测试
- 第 6 章：接口自动化测试
- 第 7 章：RF 内置测试库
- 第 8 章：持续集成自动化测试
- 第 9 章：移动自动化测试

3

第 3 章

Web 自动化测试

这一章将要开始进入到实际的应用当中，首先就是 Web 自动化测试，这个也是目前用的最多的了。在这里会介绍基于 Selenium 的 Selenium2Library（简称 S2L），还会介绍如何设计你的测试案例。

3.4 小节的测试案例：

https://github.com/qitaos/rf-demos/tree/master/rf-book-case/test

3.5 小节的测试案例：

https://github.com/qitaos/rf-demos/tree/master/Selenium2Library-demo

3.1 Selenium

在说 Selenium2Library 之前还是先说一下 Selenium，它可以算是开源 Web 测试的鼻祖

了，横跨多个浏览器、多种开发语言、多个操作系统。对于 Selenium 的概念性的内容，大家可以去百度百科看看，也可以到 Selenium 的官网（http://docs.seleniumhq.org/）看一下。

简单来说，Selenium 有 2 个版本，Selenium 1.0 和 Selenium 2.0。Selenium 1.0 也就是 Selenium RC Server，Selenium 2.0 就是加入了 Webdriver。与之对应的基于 Robot Framework 的测试库分别是 SeleniumLibrary 和 Selenium2Library，多了个“2”的就是基于 Webdriver 的。很深入的原理就不讲了。因为其实笔者一开始用的就是 SeleniumLibrary，因为它是基于 Selenium RC 的，所以很麻烦，每次都要先 Start Selenium Server，结束后再 Stop Selenium Server。并且 RC 有个不好的地方就是整个页面其实被 RC 处理过了，可能页面的 js 也被处理过了，这东西不是“回锅肉”，“过一道水”就没那么“好吃”了，所以经常会有一些表现是通过 RC 生成的页面，但是和你手动打开的页面有所差异。

幸好后面出了 Webdriver，也就是 Selenium 2.0，这回是真正的页面了，形象地说原先 RC 是在页面底层做了一些“手脚”来让你测试，这次 Webdriver 到页面上方去操作了，脱离了底层，更接近于手动的操作了。所以它对应的 Selenium2Library 也是一样的模式。

SeleniumLibrary 已经好几年没有人更新了，除非有特殊的需要，比如 Flex 测试，否则不建议使用 SeleniumLibrary 了。本书主要介绍 Selenium2Library。

3.2　Selenium2Library 安装

Selenium2Library 的官网地址：

https://github.com/rtomac/robotframework-selenium2library

在 release 下面有它的各个版本可以下载，一般推荐下载 tar.gz 包（本书写作时最新的版本是 1.6.0）。很多测试库都不是自身就能完成测试的，通常会有其他的关联的依赖，Selenium2Library 的依赖是：

```
'decorator >= 3.3.2',
'selenium >= 2.32.0',
'robotframework >= 2.6.0',
'docutils >= 0.8.1'
```

如果你安装了 Setuptools 或者 pip，基本上不用担心，因为安装的时候它会自己下载这几个库并安装好，但是如果公司的网络有限制的话，那就只能自己一个个地去找安装包下载了，通常这些安装包都可以在 Python 官网的 pypi 里下载到（https://pypi.python.org/pypi/）。

其中 Robot Framework 自然不用多说了，前面基础安装已经安装过了，其他几个如果受网络原因影响的话，请自行下载安装，地址分别如下：

https://pypi.python.org/pypi/decorator

https://pypi.python.org/pypi/selenium

https://pypi.python.org/pypi/docutils

在前面关于基础安装的章节都已经讲过了，对于这些 tar.gz 的安装包，在解压缩后，进入它的目录，输入 python setup.py install 即可（Mac 平台记得加上 sudo）。

安装完这些还不算完，如果要测试 IE，需要 IEDriver 的驱动，如果要测试 Chrome 需要 ChromeDriver 的驱动，原先这些驱动下载的地方都在 https://code.google.com/p/selenium/downloads/list 里，不过自从 google-code 要关闭了，就都转移了。IEDriver 下载的地址是 http://selenium-release.storage.googleapis.com/index.html，目前最新的版本是 2.45。进入 2.45 版本的目录，这里有 IEDriver 的 32 位和 64 位下载，根据操作系统来选择合适的版本进行下载，然后解压缩文件，将里面的 IEDriverServer.exe 文件放到 PATH 路径能访问到的地方，建议放到 Python 安装目录，因为我们已经把它加到 PATH 里了。ChromeDriver 有了一个自己正式的网站：

https://sites.google.com/a/chromium.org/chromedriver/downloads

目前最新的版本是 ChromeDriver 2.15，系统提供了 Linux 32 位、64 位和 Mac 32 位、Windows 32 位的版本，同样是下载后解压缩文件，将里面的可执行文件放到 PATH 路径，以便于能直接执行。FireFox 倒是不需要 Driver，不过它需要在 Open Browser 时指定一个 firefoxprofile，在 Selenium2Library 的源码里的 resources 目录里，有一个 firefoxprofile 的目录，大家可以根据自己的需要修改配置文件来测试。

Safari 不支持 Webdriver，只能用 Selenium RC 来启动，需要下载一个 selenium-server-standalone 的 jar 文件，然后需要在环境变量中新增一个 SELENIUM_SERVER_JAR 的环境变量，配置的值是 selenium-server-standalone.jar 的路径，然后才能启动。

除了这几个浏览器，其他支持的浏览器有 opera、phantomjs、htmlunit 等，这几个笔者只调过 phantomjs，但是没解决处理 Alert 的问题。由于笔者基本上用的不多，对这几个浏览器就不多说了，避免误导大家。

这里顺便提一下，之前在吴穹博士的支持下，我们在平安科技内部对 Selenium2Library 做了一些补充，主要是增加了一些对 WebTable，还有 BrowserHandles 的一些处理方法。想要使用这些方法的朋友可以到笔者的 github 去下载安装，地址为 https://github.com/qitaos/robotframework-selenium2library。

安装完成后，在 Import Library 的地方输入 Selenium2Library，单击“OK”按钮就可以了。如果安装正确，Library 里显示的 Selenium2Library 是黑色的就正确了。如果是红色的，可能是有些依赖库没有安装成功，或者 S2L 本身有错误，可以查看一下 RIDE LOG 找找原因。Selenium2Library 在被加载的时候也可以加上参数，它的默认参数有 timeout=5.0, implicit_wait=0.0, run_on_failure=Capture Page Screenshot，分别代表超时时间、隐式等待时间、运行失败处理。Timeout 超时时间就是每一个操作默认的超时时间是 5 秒，run_on_failure 是当运行失败的时候要做什么处理，默认是对当前页面截图。Implicit wait 可以叫做隐式等待，与之对应的就是显式等待 Explicit Wait。这两者的区别简单地说，隐式等待如果找不到对象，就会强制等待一段时间，显式等待如果找不到对象，只要没有超时，它还会继续寻找这个对象，直到找到或者超时。相比较而言，显式等待比隐式等待要好很多，具体的原因大家可以查阅一下 Selenium 官网的文章（http://docs.seleniumhq.org/docs/04_webdriver_advanced.jsp#explicit-and-implicit-waits）。

在大家下载 tar.gz 包里有一个 doc 目录，里面有 Selenium2Library 的说明文档，其中 Importing 部分介绍了如何在 Import Selenium2Library 时加上参数。

3.3　Selenium2Library 常用关键字

Selenium2Library 的关键字主要是有这些分类：browserManagement,cookie,elements（elements ,formElement,SelectElement,tableElement）,javascript,screenshot,waiting 等。分别看一下它们主要的关键字，详细的关键字列表请在 RIDE 里按 F5 快捷键查看。

3.3.1　browserManagement

这里最常用的关键字当属 Open Browser 和 Close Browser，主要用于打开和关闭浏览器。Go to 用于你要转到某个 URL，Go Back 相当于浏览器的后退。

通常做 Web 测试的时候，经常会遇到有多个窗口同时存在的情况。这里有两种情况，一种是在当前的浏览器弹出新的窗口；另一种是重新打开一个新的浏览器。

对于第 1 种情况，需要用关键字 Select Window 和 Close Window 来处理弹出的窗口，只有当选择到对应的窗口，才能对这个窗口里的页面元素进行操作。不过实际操作时会发现有时 Select Window 不能保证一次就选中那个窗口，所以通常会结合 Wait Until Keyword Succeeds 这个关键字一起使用，来保证能选中那个窗口。如果要手动关闭这个弹出窗口，可以用 Close Window。不过在通常情况下，都是在弹出窗口做了操作后，会自动关闭这个窗口，此时不要想着直接操作原来主窗口的页面元素，一定要先 Select Window Main，即回到主窗口才能操作。

对于第 2 种情况，官方给出的例子是用 2 个“Open Browser”打开了 2 个浏览器，每个浏览器分别打开不同的页面，此时两个浏览器切换需要用到 Switch Browser 这个关键字来切换，参数是打开浏览器的 index 顺序或者 alias 别名。而此时要关闭所有浏览器，最好直接用 Close All Browsers，否则如果用 Close Browser 只会关闭当前的浏览器，还要切换到另一个浏览器去关闭。

大家可以注意到，在这里提到了两个名词，浏览器和窗口，也就是 Browser 和 Window。要先弄清楚它们的区别，才不会使用错误。Browser 是通过 Open Browser 创建的，每一个 Browser 都是通过 Webdriver 创建的，你可以理解为每个 Browser 都是一个独立线程，而在当前线程的 Browser 上打开的新的页面都可以算作 Window。一个 Browser 上可以打开一个或多个 Window，具体要看页面的代码实现。每次的操作只能在一个 Window 上进行，如果当前 Window 被关闭了，必须要先切换到其他的 Window 上，才能在那个 Window 上继续操作，如果不切换，这时候的任何操作会是在一个已经关闭的 Window 上进行，此时肯定会报错。

所以 Window 切换要使用 Select Window，Browser 切换要使用 Switch Browser，不能互换混用。

在 Select Window 时，经常会遇到模态窗口，这种窗口有时不一定能直接选择到，在我们的修改版本里，添加了 3 个通过 handle 去操作窗口的关键字。即 select window by handle：通过 handle 选择 window; get window handles：获取当前线程所有窗口的 handles，返回 List; get current window handle：获取当前窗口的 handle。这 3 个关键字用好了可以帮助我们更好地处理窗口的切换。

Select Frame 和 Unselect Frame，这两个应该用的会比较多，现在很多页面都采用了 frame 或者 iframe。对于在 frame 或者 iframe 里的对象，无法直接操作，所以如果大家遇到页面元素明明就在页面里有，但是操作时却告诉你找不到页面元素的情况，首先要看的

就是这个页面元素是不是在 frame 或者 iframe 里。如果在 frame 或者 iframe 里，那就一定要用 Select Frame 先选择到这个 frame 或者 iframe，如果嵌套有好几层，就要从最外层 Select Frame，然后依次 Select Frame 到你要操作的页面元素所在的那一层，然后才能操作。操作以后，可能又要去处理 frame 外面的页面元素了，这时就要用 Unselect Frame 了，这个关键字是直接跳出所有 frame 或 iframe 的。所以对于多层 frame 或 iframe 的情况，进去的时候要一层一层地进去，出来就可以直接出来了。但是可惜没有回到上一层的，如果你的测试场景需要在 frame 或 iframe 之间切换，那就比较麻烦了。比如说这里有 3 层 iframe，你要先操作一个第 3 层的页面元素 A，然后再操作一个第 2 层 iframe 里的页面元素 B，你的脚本要先依次 Select Frame 第 1 层、第 2 层、第 3 层，然后操作你的页面元素 A，之后要操作第 2 层的页面元素 B，需要先 Unselect Frame，然后再 Select Frame 第 1 层、第 2 层，才能操作你的页面元素 B。看起来很绕吧，现在一般情况下最多 1~2 层 iframe，3 层的基本很少见到，2 层其实也不多。如果真的碰到了，最好要求开发人员仔细考虑一下此处是否真的需要那么多 iframe。

3.3.2　Cookie

Cookie 是一些网站为了辨别用户身份储存在用户本地的数据，有一些可能是涉及敏感信息，比如储存用户的用户名密码、计算机名、访问过的网站等。具体 Cookie 的作用或者其他的作用，不是本书讨论的重点。这里主要看 Cookie 的关键字都有哪些。Cookie 的关键字不多，也就下面 5 个：

- add cookie：添加 Cookie；
- delete cookie：删除 Cookie；
- delete all cookies：删除所有 Cookie；
- get cookies：获取所有 Cookie；
- get cookie value：获取 Cookie 的值。

如果你测试的网站有需要对 Cookie 进行操作的话，请使用上述的关键字进行处理。

3.3.3　Elements

从操作 Elements 开始，要了解一个新的东西，叫 locator。它主要是用来定位页面元素的，很多对页面元素的操作都需要先定位到它才能操作。比如要输入文本信息到一个文本框，locator 就是告诉 Selenium 要对哪个对象进行操作了。常见的 locator 有 id、name，因为 id 是最方便的，所以之前要求开发人员必须给每个元素加上 id，如果没有 id，可以上报

bug，算是一种强制手段。如果没有 id 的话，可以用 name，但是 name 是可以同名的，可以有多个页面元素使用同一个 name，这就给定位带来了困难。目前 S2L 在重名元素的处理上还有许多欠缺，也只能用其他手段来处理。比如 xpath,css,jQuery,sizzle 等，也可以用 DOM,HTMLtag 等，但是这些都不如 id 和 name 使用方便。

1. Elements

常用关键字有 click element,click link,click image，这几个点击事件应该不用多说了，press key 是模拟键盘操作的，在某些时候如果点击不好用的话，也可以用“press key \\13”模拟按“回车”键来代替点击。

Assign id to element 也有用处。一个对象如果没有 id，你可以给它指定一个 id，这样以后就不用写冗长的 locator 了

除了元素的点击操作以外，还有很多鼠标操作，比如 mouse down、mouse up、mouse over、mouse out 等。笔者更倾向于用 AutoItLibrary 的鼠标操作。

Get horizontal position、get vertical position 是用来获取当前对象的左上角顶点距离整个页面的横坐标和纵坐标。

Get value、get text 用来获取网页元素的 value 或者 text，使用时要看清楚网页源码对象是不是有 value 或者 text。如果它没有 text 去 get text，肯定会取到空值。如果想获取元素其他属性值，要用 get element attribute 关键字，比如要获取 class 值，要用 get element attribute element_id@class 来获取。

2. formElement

form 也就是表单，虽然这里起名是 formElement，其实里面有很多基本对象的操作，包括 checkbox,radio buttons,text fields,button。当然，它们几个放在这里也可以理解，因为它们的 HTML 源码都是 input 标签的，而且都是表单里所需要使用的。

form 直接有关的就一个关键字 submit form，用于提交表单的。

接下来是 checkbox 和 radio buttons。checkbox 主要是 select checkbox 和 unselect checkbox，作用是勾选复选框和不勾选复选框。Radio buttons 主要关键字是 select radio button。其他的一些关键字这里不进行细说了。

text fields 部分常用的有 choose file、input text、input password。choose file 用于上传文件的时候选择文件；input text 和 input password 其实本质上一样，只不过 input password 在输入密码的日志上不会把密码打印出来。

Button 最常用的是 click button，点击按钮即可。

3. SelectElement

SelectElement 主要是介绍列表框，列表框分单选和多选。选择列表框中的某一项，需要使用 select from list，如果你想选择得更快一些，可以使用这 3 个关键字：select from list by index、select from list by value、select from list by label，分别是通过 index/value/label 来选择列表项。

如果是多选列表框想要全部选择可以用 select all from list。有选中同样也有对应的 unselect，只有多选列表框才能 unselect。unselect from list、unselect from list by index、unselect from list by value、unselect from list by label 都是用来取消选择的。

如果你想知道列表框里都有什么列表项，可以用 get list items 来获取所有的列表项。

如果想确认一下选中的列表项对不对，可以用 get selected list label 和 get selected list value。如果是多选列表框，这两个关键字默认取第 1 个列表项。如果想要取到所有的选中项，就把这两个关键字后面的 label 和 value 变成 labels 和 values。

4. tableElement

对于 table 来说，是 Web 页面里最多出现的对象了，很多网页布局都用到了 table。原版的版本里最有用的关键字就是 get table cell 了，可以获取指定 table 里的指定行列的元素的文本值。其他的关键字多数都是用来判断元素是否存在的。

吴穹博士在平安科技指导工作的时候，帮助我们增加了一些关键字，比如 click element at table cell 和 click link at table cell 用于点击表格指定单元格里的元素和链接。

当然笔者觉得更有用的是 get index in table column 和 get index in table row，分别用来获取某个文本在表格的哪一行或哪一列。

由于笔者之前用 QTP 比较多，QTP 里 Webtable 提供的获取表格行数和表格列数的方法觉得比较实用，所以后来笔者补充了两个关键字，get table rows 和 get table cols at row，分别用来获取表格的行数和在某一行表格有多少列。

添加这些关键字主要的原因还是因为 tableElement 里面提供的用于操作 table 元素的方法太少了，只有一个 get table cell 是操作 table 元素的，但是这里需要指定 table 的行和列，实际使用的时候你会发现很多场景都是我们并不知道我们要的数据在哪行哪列，如果只知道行或列，那么吴穹博士提供的 get index 的两个关键字就会发挥作用；如果行列都不知道

（好像不太可能出现）或者你想遍历 table 里所有单元格，那么用笔者加的那两个关键字就会比较有用。

3.3.4 JavaScript

JavasScript 这一部分主要提供了 execute JavaScript 和 execute async JavaScript 这两个关键字，都是用于执行 JavaScript 脚本，后者是用于异步执行 JavaScript 脚本。异步执行 Javascript 通过在函数最后加上回调来返回结果，如果在超时时间内没有执行完成，那么就会失败。

除了 JavaScript，这部分最有价值的关键组当然是 confirm action。在网页上经常会有一些 alert 或者确认对话框弹出，用 confirm action 会自动进行对话框的确认，如果不对对话框进行处理，我们的案例可能就会在这里卡住。确认对话框通常有确认和取消，默认是确认。如果你想让 confirm action 点取消，那就要先用 choose cancel on next confirmation 进行设置，然后再使用 confirm action。注意 choose cancel 这个关键字是做一个设置，并不是真的去确认对话框，确认对话框还是 confirm action 要做的事情。同样你想恢复成默认点“ok”，就需要用 choose ok on next confirmation 进行设置，然后再使用 confirm action。

3.3.5 screenshot

screenshot 只有一个关键字 capture page screenshot，用于对当前页面进行截图。但是这里有个问题，它的截图是基于当前浏览器的页面进行截图的，如果此时由于某些报错导致页面被关闭了，截图就根本没法使用了。正因为我们碰到了这样的情况，因此笔者将这个截图的方法稍作修改，让它使用 RF 自带的截图库里的方法进行截图，而那个截图是对当前整个桌面进行截图的，所以即使浏览器关闭了，我们也能看到截图。

3.3.6 waiting

waiting 里的关键字不多，主要就 3 个，即 wait for condition、wait until page contains、wait until page contains element，这些其实就是前面提到的显式等待了，它们会持续尝试条件是否满足直到超过时间。其实笔者用的 wait for condition 不多，更多情况下还是更愿意用 BuiltIn 里的关键字 Wait Until Keyword Succeeds，而 wait until page 的两个关键字在页面加载比较慢的时候还是很有用的，避免我们在页面没有加载完就先去操作页面元素。鼓励大家多用显式等待。

3.4　测试案例设计

本小节来说一下测试案例设计。首先，从官网下载的 robotframework-selenium2library-1.6.0.tar.gz 文件（每个版本都有，可以使用最新的），在解压缩后，有一个 demo 目录，通过命令行进入这个目录，输入命令来启动 demo 服务：

```
python rundemo.py demoapp start
```

要停止的话请把“start”换成“stop”。启动 demo 服务后，可以访问一下 http://127.0.0.1:7272/html/。来看一下这个页面，如图 3-4-1 所示。

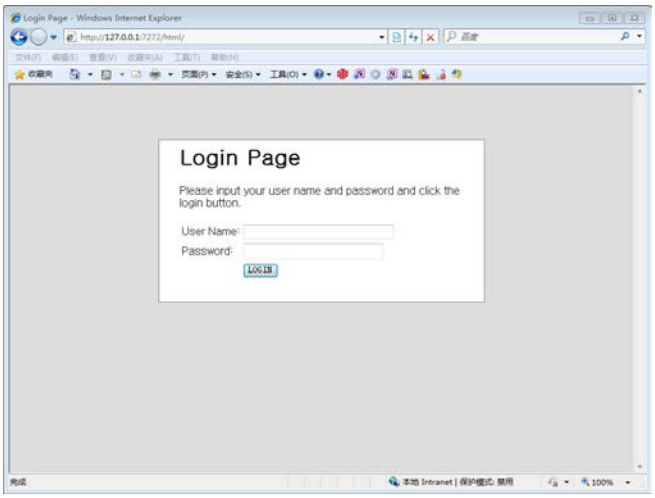

图 3-4-1

可以看到这是个登录页面，需要输入 User Name 和 Password，还有个“LOGIN”按钮，需要查看一下这些网页元素的 html 源代码，以便于找到它的关键属性。如果用 IE 浏览器，可以按 F12 快捷键打开开发人员工具，单击“箭头”，再单击要查看的元素，就可以看到对应的源码了。如果用 Chrome 浏览器，可以在元素上单击鼠标右键审查元素，都比较方便。

完整的网页源码如下：

```
<html>
<head>
  <title>Login Page</title>
  <link href="demo.css" media="all" rel="stylesheet" type="text/css" />
  <script type="text/javascript">
    function login() {
       if (document.login_form.username_field.value == 'demo' &&
```

```
            document.login_form.password_field.value == 'mode') {
            return true
         }
         else {
            window.location="error.html"
            return false
         }
      }
    </script>
  </head>
  <body>
    <div id='container'>
      <h1>Login Page</h1>
       <p>Please input your user name and password and click the login
button.</p>
       <form name='login_form' action="welcome.html" method="post" onsubmit=
"return login()">
         <table>
           <tr>
             <td>User Name:</td>
             <td><input name="username_field" size="30" type="text" /> </td>
           </tr>
           <tr>
             <td>Password:</td>
             <td><input name="password_field" size="30" type="password" /> </td>
           </tr>
           <tr>
             <td> </td>
             <td><input name="login_button" type="submit" value="LOGIN" /> </td>
           </tr>
         </table>
       </form>
    </div>
  </body>
  </html>
```

可以看到一些关键的信息，比如用户密码是 demo/mode，User Name、Password 和 LOGIN，这 3 个元素的 name 分别是 username_field、password_field、login_button。

3.4.1 案例设计 Step1

先设计一个简单的手工案例，操作步骤如下：

（1）打开浏览器显示 demo 页面；
（2）输入用户名 demo；
（3）输入密码 mode；
（4）单击“LOGIN”按钮；
（5）在登录成功的页面检查是否有“Login succeeded”字样；
（6）关闭浏览器。

那么我们创建一个目录型的 Suite-3-4，下面创建 Step1 的 suite，加载 Selenium2Library，新增一个 Case1。测试案例的脚本见表 3-4-1。

表 3-4-1

Open Browser	http://127.0.0.1:7272/html/	ie
Input Text	name=username_field	demo
Input Password	name=password_field	mode
Click Button	name=login_button	
Page Should Contain	Login succeeded	
Close Browser		

运行案例，运行结果如图 3-4-2 所示。

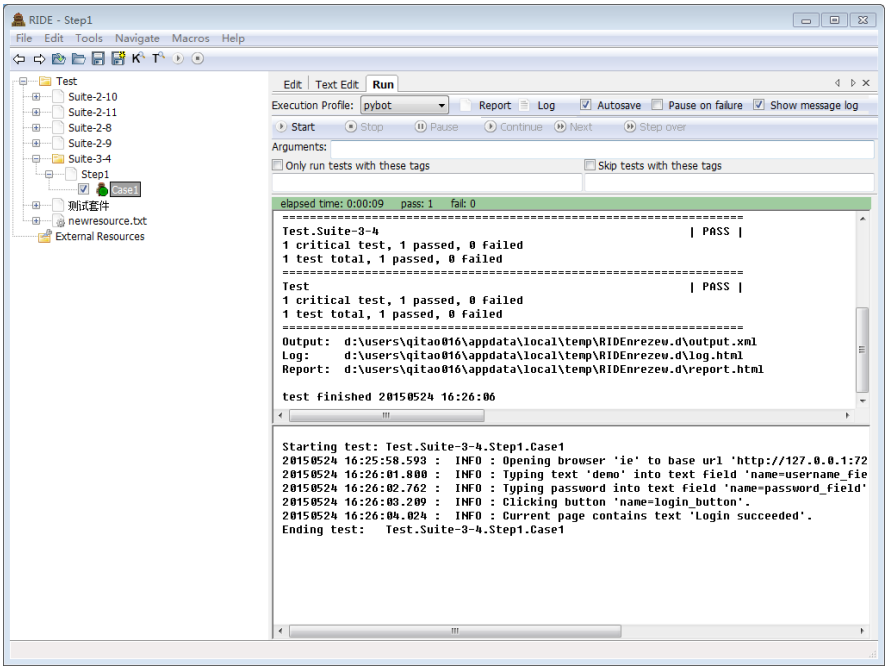

图 3-4-2

3.4.2 案例设计 Step2

可以看到这样的案例都是英文的，有点不太符合大家平时的阅读习惯，所以稍微做一点调整。为了方便大家看到变化，新增一个 Step2 的 suite，复制 Case1 然后改名为 Case2，在 Suite-3-4 下新增一个资源文件 elements.robot，在资源文件里新增一些用户关键字，将 Case2 里的脚本进行一次封装。比如新增一个关键字“打开浏览器”，给个参数${url}。关键字里实际的脚本见表 3-4-2。

表 3-4-2

Open Browser	${url}	ie

接下来把 Input Text 也这样处理，加一个关键字“输入用户名”，只不过只定义一个参数${text}就够了，Input Text 中的 locator 是不需要的，也不推荐做参数，这样后面出现了错误的时候就直接知道是哪个 locator 出问题了。关键字里的实际的脚本见表 3-4-3。

表 3-4-3

Input Text	name=username_field	${username}

接下来把每一行都这样处理，最终在案例 Step2 所在的 suite 上加载资源文件 elements.robot，资源文件上加载 Selenium2Library，把 Case2 里的脚本都换成关键字，最后脚本见表 3-4-4。

表 3-4-4

打开浏览器	http://127.0.0.1:7272/html/	
输入用户名	demo	
输入密码	mode	
点击登录		
验证页面文本	Login succeeded	
关闭浏览器		

这回我们的案例是不是和最开始我们设计的案例步骤比较类似了呢？

前面是一个登录成功的例子，那么复制 Case2，做一个登录失败的案例 Case3，其实步骤都是一样的。看一下脚本，见表 3-4-5。

表 3-4-5

打开浏览器	http://127.0.0.1:7272/html/	
输入用户名	demo	
输入密码	wrong	
点击登录		
验证页面文本	Login failed	
关闭浏览器		

运行结果如图 3-4-3 所示。

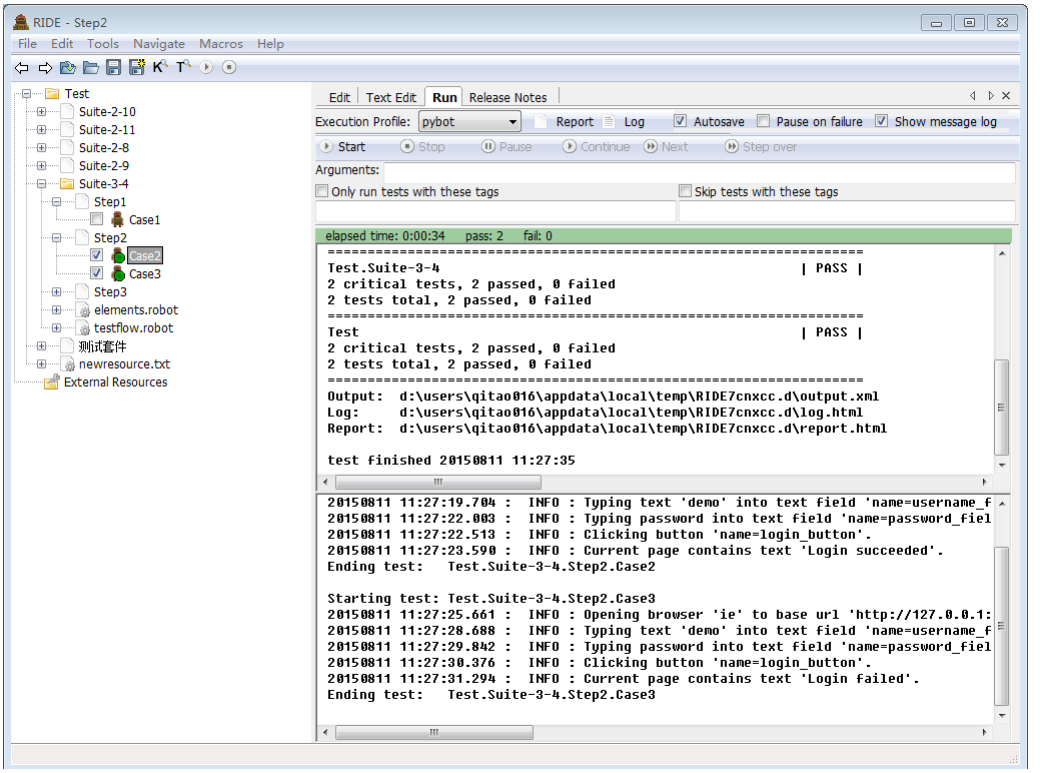

图 3-4-3

在设计测试案例的时候，通常会从正向案例和反向案例去设计。对于这个场景来说，还可以设计出很多个失败的场景，比如用户名为空，密码为空等。假设最后可能有 10 个案例，再假设这个时候，流程发生了变更，需要先输入密码，再输入用户名（虽然在这个案例里，这个流程变化其实没有什么影响，但是这里只是举例子说明一下需求变更的影响），

那么 10 个案例可能就需要修改 10 次。如果有更多的案例，估计修改的工作量会更大。那么此时有一个比较好的建议就是对测试案例进行分层。

其实前面已经算是分了两层了，只不过根据实际的情况，建议一般分三层是比较合适的，四层也可以考虑，更多分层就不建议了。

3.4.3 案例设计 Step3

接下来同样像 Step2 那样做，再增加一个 Step3 的 suite，Case 都复制过去，然后在 Suite-3-4 下面再新增一个资源文件 testflow.robot，这里面来放测试流程。然后调整一下资源加载的顺序，原先的测试套件加载资源文件改成 testflow.robot，然后 testflow.robot 加载资源文件 elements.robot。

对我们目前的场景来说就一个流程，我们给 testflow 里的关键字起个名字叫“登录验证流程”，把流程里的数据剥离出去，通过关键字的参数来传递，因此这个关键字会有几个参数${url}|${username}|${password}|${text}。关键字的脚本见表 3-4-6。

表 3-4-6

打开浏览器	${url}	
输入用户名	${username}	
输入密码	${password}	
点击登录		
验证页面文本	${text}	
关闭浏览器		

然后在案例里就是直接调用流程的关键字，脚本见表 3-4-7。

表 3-4-7

登录验证流程	http://127.0.0.1:7272/html/	demo	mode	Login succeeded

另一个案例的脚本类似，只是参数值换了一下，脚本见表 3-4-8。

表 3-4-8

登录验证流程	http://127.0.0.1:7272/html/	demo	wrong	Login failed

再运行一下案例看看，运行结果如图 3-4-4 所示。

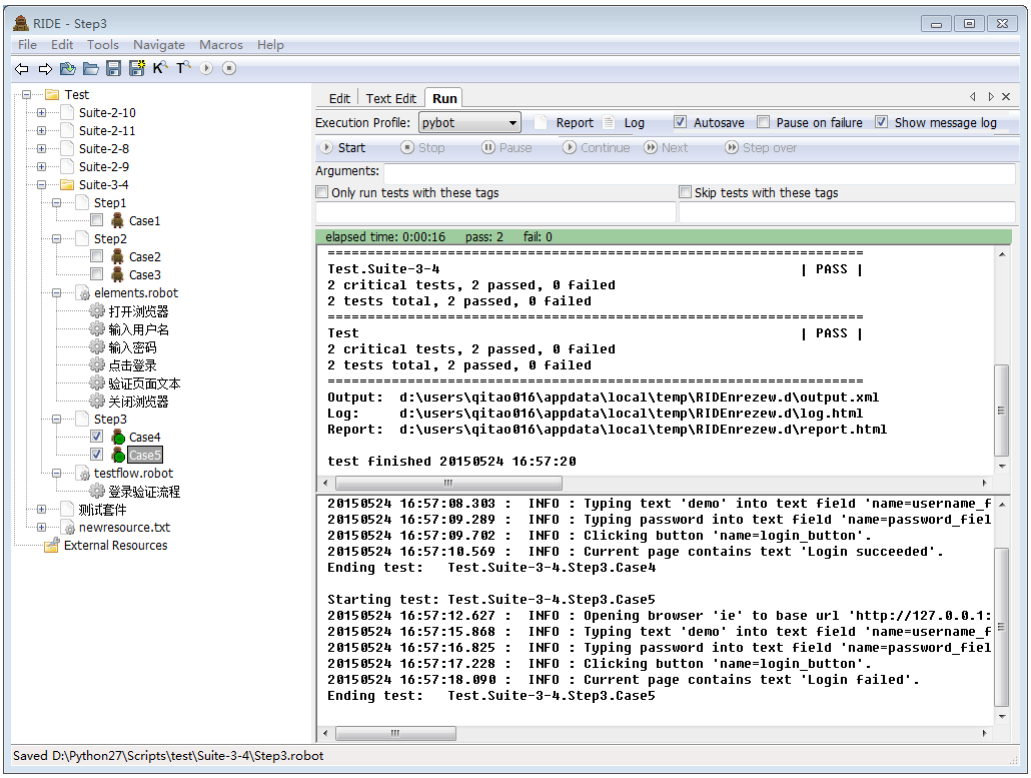

图 3-4-4

此时我们只是分了三层，三层的结构见表 3-4-9。

表 3-4-9

案例层	测试流程+数据	
流程层	元素操作	例如打开浏览器、输入用户名
元素层	测试库封装	Open Browser封装成打开浏览器

这样做有什么好处呢？如果还是刚才那个需求变更，只需要在流程层里调整流程的顺序就可以了，而不用把每个案例都修改一次。

刚才也提到了可以分四层的情况，这时候可以引入 PageObject，把页面上的操作整合在一起，也会比较方便。分四层的结构见表 3-4-10。

表 3-4-10

案例层	测试流程+数据	
流程层	页面层操作	页面1的操作、页面2的操作
页面层	元素操作	例如打开浏览器、输入用户名
元素层	测试库封装	Open Browser封装成打开浏览器

最终经过分层，把数据和流程剥离开，同时减少了冗余，能够减少一些由需求变更引发的修改。建议大家根据自己的实际情况将案例结构分成三层或四层。数据可以进一步剥离，比如说将数据存在变量文件、数据库或者 Excel 里，通过运行的时候去读取数据来运行。

3.5 测试案例 demo

3.5.1 使用 demo 前的准备

前面的案例是用来介绍分层的，使用的是官方的 demo 例子。为了保证后续的例子能达到期望的效果，我用了 flask 来实现了一个 demo 网站，它可以快速建立起一个网站，只用一些简单的语句。

如果使用 pip 安装会比较方便，特别是在 Mac 电脑上。为了保证例子能正常运行，需要安装 4 个东西，flask、flask-httpauth、flask_restful、python-simplexml。

只需要运行如下命令：

```
#mac
sudo pip install flask
sudo pip install flask-httpauth
sudo pip install flask_restful
sudo pip install python-simplexml
#windows
pip install flask
pip install flask-httpauth
pip install flask_restful
pip install python-simplexml
```

如果你的 pip 不好用或者网络受限制，那么只能下载单独的源码包进行安装。

下面这些都可以在页面（https://pypi.python.org/pypi/）上面找到，而且甚至不用搜索，

直接在 pypi 后加上库名字即可。例如 flask，它的下载地址是 https://pypi.python.org/pypi/flask。下载源码后解压缩，进入解压缩目录，输入命令：

```
    #mac
    sudo python setup.py install

    #windows
    python setup.py install
```

下面列的这些都是需要下载手动安装的（缩进是表明依赖关系）。如果觉得很麻烦还是去做 pip 安装吧：

```
flask
    Werkzeug
    Jinja2
       MarkupSafe
    itsdangerous
    click
flask-httpauth
flask_restful
    aniso8601
    six
    pytz
python-simplexml
```

安装完成之后就可以运行我们的 demo-website 了。本书全部的 demo 的地址是：

https://github.com/qitaos/rf-demos

具体说明可以看网站上的 README，所有章节的 demo 都会放到这里，同时书籍出版后也会继续更新，所以你看到 github 上 rf-demos 的内容可能会比书上的多。

demo-website 的地址：

https://github.com/qitaos/rf-demos/tree/master/demo-website

下载整个工程之后，进入 demo-website 目录，运行命令：

```
python flaskdemo.py
```

运行成功后会有提示信息：

```
> python flaskdemo.py
```

```
 * Running on http://127.0.0.1:8000/ (Press CTRL+C to quit)
 * Restarting with stat
```

这里提示我们服务器已经启动，访问 http://127.0.0.1:8000/ 或者 http://localhost:8000/ 就可以打开页面了。

如果有任何出错信息，有可能是前面的测试库没有安装完整。

3.5.2 Selenium2Library-demo

demo 的名字就是对应的目录名，所有的 Case 都在里面。

按照前面的关键字模块，分别列举一些涉及不同模块的例子。这里的案例都是在 Windows 7 操作系统下面用 IE 浏览器执行的，如果你是使用的其他浏览器，请根据自己的需要修改案例。

1. Browser

（1）切换浏览器

这个案例主要是为了演示 Switch Browser 的作用。前面其实也提到了，通过 Open Browser 打开的两个浏览器可以用 Switch Browser。Switch 之后通过 Get Title 获取当前浏览器的 Title，来确认是否切换成功。因为开了多个 Browsers，所以最后要用 Close All Browsers 关掉所有的 Browsers。如果用 Close Browser 只会关闭当前的 Browser，感兴趣的朋友可以自己尝试。脚本见表 3-5-1。

表 3-5-1

Open Browser	http://localhost:8000/	ie	local
${title1}	Get Title		
Open Browser	http://www.baidu.com	ie	baidu
${title2}	Get Title		
Switch Browser	local		
${title1}	Get Title		
Switch Browser	baidu		
${title2}	Get Title		
Close All Browsers			

（2）选择窗口

这个案例主要是为了演示在一个 Browser 上，弹出 Alert 用 Confirm Action 处理，弹出新的窗口用 Select Window 处理，Select Window 之后，你才能对那个新的窗口里的对象操作，这里笔者只是用了一个 Log Source 打印新窗口的 HTML 源代码。虽然此时有两个窗口，但是它们都是一个 Browser 产生的，所以用 Close Browser 就可以都关掉了。脚本见表 3-5-2。

表 3-5-2

Open Browser	http://localhost:8000/	ie
Click Button	id=pay	
Confirm Action		
Select Window	付款	
Log Source		
Close Browser		

（3）选择 iframe

这个案例主要演示打开浏览器后，要操作页面上的一个 iframe 里的对象。先 Select Frame 选择到这个 id=fra 的 iframe，然后获取它里面的一个 id=buy 的下拉列表的所有元素。之后用 Unselect Frame 跳到 iframe 外面，然后就可以操作外面的对象了，这里用 Log Source 打印了一下 HTML 源代码，脚本见表 3-5-3。

表 3-5-3

Open Browser	http://localhost:8000/	ie
Select Frame	id=fra	
${list}	Get List Items	id=buy
Unselect Frame		
Log Source		
Close Browser		

2. Elements

（1）文本框操作

这个案例主要演示文本框的操作。主要是用 Input Text 去输入文本，然后再用 Get Value 来获取值（你可以用这个值验证一下是否录入正确）。接下来笔者用了一个关键字 Assign Id To Element，一般推荐大家使用 id 来定位对象，但是有时候开发人员漏了没写，我们可

以用这个关键字来给这个对象指定一个 id，你可以用其他的定位方法来定位，比如这里就用了 name 来定位。之后就可以用你指定的 id 来对这个对象操作了。脚本见表 3-5-4。

表 3-5-4

Open Browser	http://localhost:8000/	ie
Input Text	id=ProductName	新产品
${value}	Get Value	ProductName
Assign Id To Element	name=Quantity	Quan
Input Text	id=Quan	111
${qn}	Get Value	Quan
Close Browser		

（2）按钮操作

这个案例主要演示按钮的操作。它的操作其实很简单，操作 Click Button 就行了。如果你想获取 Button 上的文字，可以用 Get Text 来获取。脚本见表 3-5-5。

表 3-5-5

Open Browser	http://localhost:8000/	ie
Input Text	id=ProductName	新产品
Input Text	id=Quantity	111
Click Button	id=submitBtn	
${text}	Get Text	infoBtn
Close Browser		

（3）获取页面元素值

这个案例来演示如何获取页面元素值。其实前面的例子里已经有一些了，比如用 Get Value 来获取文本框的值，用 Get Text 获取按钮上的文本信息。这些都是网页对象的一些基本属性，如果你想获取其他的属性值怎么办呢？Get Element Attribute 这个关键字可以帮助我们，不过这里的写法有点区别，在关键字说明（按 F5 快捷键）里有介绍，它的参数 attribute_locator 是需要用 element_id@class 这样的形式，简单说就是“@”前面是这个元素的 locator 定位，“@”后面是你想要获取的属性名。比如我们案例里就用了 Pr@innerText 来获取 id=Pr 对象的 innerText 值，用 ProductName@name 来获取 id=ProductName 对象的 name 值。脚本见表 3-5-6。

表 3-5-6

Open Browser	http://localhost:8000/	ie
Input Text	id=ProductName	新产品
${value}	Get Value	ProductName
Input Text	id=Quantity	222
Click Button	id=submitBtn	
${text}	Get Text	infoBtn
${text}	Get Element Attribute	Pr@innerText
${name}	Get Element Attribute	ProductName@name
Close Browser		

（4）下拉列表

这个案例主要演示下拉列表是如何操作的。默认我们使用 Select From List 来选择下拉列表的某一项，实际上它还有更具体的 3 个关键字，就是 Select From List By Index、Select From List By Value、Select From List By Label。这 3 个关键字分别是怎么使用的呢？看一下这里的下拉列表的源代码：

```
   <select id="selectdemo" name="selectdemo"><option value="item1">第一个元素</option><option value="item2">第二个元素</option><option value="item3">第三个元素</option></select>
```

首先 By Index 很好理解，它是通过序列 Index 来选择元素，但是要记住第 1 个元素的 Index 是 0，By Value 就是源代码里 option 的 value 值，所以案例里有通过 item2 的 Value 来选择的，而 Label 是显示在下拉列表里你能看到的列表项的文字，比如第 3 个元素。这里没有演示 By Label 的步骤，不过用 Get Selected List Label 来进行取值，让大家知道 Label 值是什么。如果默认使用 Select From List，这个关键字会先判断后面的值在不在列表项内，如果不在就认为是用 Index 来选择，如果在列表项里，就会先通过 Value 来选择，如果找不到则再会用 Label 来选择。可以看出来这样多了几个判断，实际上选择起来就有些慢了。所以，如果你知道自己这个选择的值是什么，最好通过 By Index/By Value/By Label 来直接选择，这样速度会更快些。脚本见表 3-5-7。

表 3-5-7

Open Browser	http://localhost:8000/	ie
Select From List	selectdemo	第3个元素
${label}	Get Selected List Label	selectdemo
Select From List By Index	selectdemo	0

续表

${value}	Get Selected List Value	selectdemo
Select From List By Value	selectdemo	item2
${label}	Get Selected List Label	selectdemo
Close Browser		

（5）其他元素操作

这个案例介绍其他一些元素操作的演示。对于 Checkbox 复选框来说，你可以用 Click Element 来点击，也可以用 Select Checkbox 或者 Unselect Checkbox 来进行操作，后面两个比第 1 个要好些，因为你明确知道你是在 Select 还是 Unselect，用第 1 个的话你只是知道点了，但是是选中还是未选中就不知道了，可能还要去取值判断才知道。

除了 Click Element，还可以用 Press Key 来操作，“\\13”代表回车，这里就是在那个对象上按回车，其实和点击效果是一样的。如果有些时候遇到点击不了的对象，不妨试试 Press Key。

说完了复选框，自然还有单选按钮 Radio Button。选择单选按钮的时候可以通过 Select Radio Button 来进行选择，也可以用前面的 Click 来选择，这里当前就是明确的点击选中了。

最后又加了个 Click Link 的例子，很多人不知道 Link 怎么点击，其实很简单，只要后面写 Link 的文本就行了。脚本见表 3-5-8。

表 3-5-8

Open Browser	http://localhost:8000/	ie
Click Element	CheckYes	
Select Checkbox	CheckNo	
Unselect Checkbox	CheckYes	
Press Key	CheckNo	\\13
${yes}	Get Value	CheckYes
Select Radio Button	radio1	B
${radio}	Get Element Attribute	radioB@checked
${value}	Get Value	radioB
Click Link	上传下载demo页面	
Sleep	2s	
${title}	Get Title	
Close Browser		

（6）表格元素操作

这个案例主要演示表格元素的操作。其实官方的表格元素的操作比较少，或者说就只有一个 Get Table Cell 的关键字是取值的，其他关键字都是去做判断的。虽然改造了一下，加了一些方法，但是为了避免有些朋友没有安装修改版无法运行案例，这里只列出了官方的关键字。除了 Get Table Cell 以外，剩下的关键字就是“Table XXX Should Contain”，这个“XXX”有很多，有 Cell,Column,Row,Footer,Header 等，这里只举了个 Cell 的例子。当然不加其他几个关键字的主要原因是在 IE 8 浏览器上判断 Row 和 Column 总是有问题的，原因主要由于 IE 8 浏览器对 xpath 和 css 支持的不规范导致，在 Chrome 浏览器上是正常的。脚本见表 3-5-9。

表 3-5-9

Open Browser	http://localhost:8000/	ie		
${cell}	Get Table Cell	buy	1	1
${cell}	Get Table Cell	buy	1	2
Table Cell Should Contain	buy	1	1	产品
Table Cell Should Contain	buy	1	2	数量
Close Browser				

3. Others

（1）cookie 操作

这个案例主要演示如何操作 cookie。当然笔者的 demo 网站没加 cookie，所以值第 1 次 Get Cookies 的时候是空的。然后我们用 Add Cookie 来手动添加 cookie，第 2 次 Get Cookies 的时候就有了刚刚添加的内容了。如果想获取其中一个 cookie 的值，那就要用 Get Cookie Value 来获取。脚本见表 3-5-10。

表 3-5-10

Open Browser	http://localhost:8000/	ie
${co}	Get Cookies	
Add Cookie	test	qitao
Add Cookie	robot	framework
${co}	Get Cookies	
${value}	Get Cookie Value	test
Close Browser		

（2）JavaScript 操作

这个案例主要演示 JavaScript 的操作，主要是用 Execute JavaScript 来执行脚本。笔者用了第 1 个 JavaScript 修改了一下 infoBtn 的只读变成 false，之后的一些案例都是用的 jQuery 来执行的。只要都是 JavaScript 的脚本，都可以拿来执行。除了 Execute JavaScript 外，还有一个 Execute Async JavaScript 用于执行异步的 JavaScript 脚本，这里没有准备它的例子，大家感兴趣可以自己尝试。脚本见表 3-5-11。

表 3-5-11

Open Browser	http://localhost:8000/	ie
Execute Javascript	document.getElementById("infoBtn").disabled=false	
Execute Javascript	$("#ProductName").val("jquery")	
Execute Javascript	$("#Quantity").val("123")	
Execute Javascript	$("#submitBtn").click()	
Sleep	5s	
${pr}	Execute Javascript	return $("#Pr").html()
${qn}	Execute Javascript	return $("#Qn").html()
Close Browser		

（3）截图

这个案例主要演示截图这个关键字。就这一个关键字 Capture Page Screenshot，参数是文件名，如果写了文件名就会每次都以这个文件名保存，如果存在就会覆盖。如果不写文件名，则会默认生成一个带序号的文件名，截图文件默认是保存在 Log 文件输出目录。

之前笔者对这个方法做了修改，主要是防止 Browser 关闭后无法截图。其实原生的截图也是挺好用的，它是截取网页 Page 的全屏，如果你是在 iframe 里，它的截图就是 iframe 那个区域的图。脚本见表 3-5-12。

表 3-5-12

Open Browser	http://localhost:8000/	ie
Capture Page Screenshot		
Select Frame	id=fra	
Capture Page Screenshot	iframe.png	
Close Browser		

（4）waiting

其实这个案例内容和前面的 JavaScript 的案例基本一样，笔者只是把 Sleep 换成了 Wait

Until Page Contains。在 Selenium2Library 里 Wait 开头的关键字有 6 个，它们的作用都是大同小异，就是在超时时间内，等到满足条件为止，其中 5 个是明确写了什么条件。比如我们用的这个 Page Contains，还有 Element Is Visible 等，如果这些不够，还可以用 Wait For Condition 来写条件。

这里介绍 Wait 相关的内容，主要是网页加载有快有慢，但是案例脚本执行是比较快的，有时候加载快的话案例执行成功了，加载慢的话案例就失败了，因为页面对象还没加载出来，你的脚本就去操作它了，当然会失败。那么这时候如何处理呢？这就用到了我们说的 wait 的一系列关键字来进行同步，同步的意思是让网页的页面和案例的脚本同步，如果不同步很容易就会失败。同步的方式，常用的有两种，一种就是 Sleep，这个是“硬等”，不管中间页面加载情况如何，它也会安装你设置的时间“Sleep”过去；另一种就是通过条件判断的 wait 关键字了。后者比前者要好很多，比如你用 Sleep 10s（等待 10 秒），这时候案例就会等 10 秒，而如果你设置 wait 的超时为 10 秒，那么假设 4 秒的时候网页加载出需要的对象了，这时候 wait 的工作就做完了，交给后面的步骤去执行了，这样后者实际只等了 4 秒，比 Sleep 更高效。脚本见表 3-5-13

表 3-5-13

Open Browser	http://localhost:8000/	ie
Execute Javascript	document.getElementById("infoBtn").disabled=false	
Execute Javascript	$("#ProductName").val("jquery")	
Execute Javascript	$("#Quantity").val("123")	
Execute Javascript	$("#submitBtn").click()	
Wait Until Page Contains	123	
${pr}	Execute Javascript	return $("#Pr").html()
${qn}	Execute Javascript	return $("#Qn").html()
Close Browser		

3.6　常见问题

前面介绍了常用的一些关键字，下面说说一些容易碰到的问题。

（1）不要修改浏览器的显示比例，如果不是 100%，可能会导致运行失败。

（2）使用 IE 打开浏览器时报错：“ IEDriver executable needs to be available in the path. Please download from http://selenium-release.storage.googleapis.com/index.html”，这就是我

们要下载 IEDriver 并将其放在 PATH 里，这样才可以执行（Chrome 浏览器也是需要 ChromDirver 的）。

（3）打开 IE 浏览器时另一种报错：“WebDriverException：Message：u'Unexpected error launching Internet Explorer. Protected Mode settings are not the same for all zones. Enable Protected Mode must be set to the same value (enabled or disabled) for all zones.”。这个错误是说我们要在 IE 选项的“安全”选项卡里面，在 4 个区域里都要设置一下保护模式，要保持 4 个区域是一致的，都勾选复选框或者都不勾选。

（4）定位不到网页元素，最容易出错的就是网页元素在 iframe 里，一定要先 Select Frame，之后再进行操作。

3.7 小结

本章主要介绍了 Web 自动化测试的一些例子。基于 Web 的自动化测试通常最多需要考虑的就是 wating，如何合理的添加等待，可以减少因为页面加载慢导致的错误，也可以让测试案例的时间更合理。另外介绍了如何进行测试案例设计，分层的思路，希望大家合理利用分层，能够减少一些更新维护的工作量。

4

第 4 章

C/S 自动化测试

本章主要介绍 AutoItLibrary 测试库的安装、如何开展 C/S 的测试，以及结合 Selenium2Library 进行的测试。本章仅适用于 Windows 平台。

本章的测试案例：

https://github.com/qitaos/rf-demos/tree/master/AutoItDemo

4.1 AutoIt

AutoIt 是一个用于对 Windows 图形界面进行自动化操作的软件，现在官网上是 AutoIt v3 版本，它可以模拟键盘操作、鼠标操作、窗口和控件操作来进行自动化。因为是针对 Windows 平台设计的，所以在 Mac 上就不用想了。另外，这个也只能针对标准的 Windows 32 位操作系统的对象进行操作，如果是一些自定义对象的话，很难用 AutoIt 来做自动化。

4.2 AutoItLibrary 安装

针对 AutoIt，有人开发了一个 AutoItLibrary 测试库，基于 Robot Framework 框架，不过目前已经很久没有更新了。

1. 安装 pywin32

http://sourceforge.net/projects/pywin32/files/pywin32

在安装 AutoItLibrary 之前一定要先安装 pywin32，最新的版本是 219，请选择对应 py27 的 32 位版本，如果你的操作系统是 64 位的，请选择对应 py27 的 64 位安装包。

2. 安装 AutoItLibrary

官网地址：

https://code.google.com/p/robotframework-autoitlibrary

如果大家担心没有办法访问谷歌，笔者把它导出了一份到笔者的 github 了，地址是：

https://github.com/qitaos/robotframework-autoitlibrary

安装时注意两点，第 1 点是要有管理员权限，因为要注册一个 autoit 的 dll 文件；第 2 点是 Python 安装目录不能有空格，如果有空格，在注册 dll 的时候会报错。如果 Python 安装目录有空格，此时请卸载重装 Python 吧。

AutoItLibrary 版本就是 1.1 了，下载了 tar.gz 包后，用 python setup.py install 来安装即可。

3. AutoIt

对于操作系统是 32 位的朋友来说，前面两步就够了，如果是 64 位操作系统的朋友，此时还要安装一下 AutoIt，不是 AutoItLibrary。下载地址是：

https://www.autoitscript.com/site/ autoit/downloads/

主要是因为 AutoItLibrary 里面带的 dll 是 32 位的，无法在 64 位操作系统上使用，只能安装官方的 AutoIt，让它的 dll 生效，这样就能在 64 位操作系统上使用了。

4.3　AutoItLibrary 关键字

1. 关键字

AutoItLibrary 的关键字大体上有几大主要部分，Window 操作、Control 操作、Mouse 操作、Process 操作、Run 操作、Reg 操作，还有一些其他的操作。

- Window 操作：你看到的窗口就是 Window，所有的操作都首先要基于窗口，如果有多个窗口，则需要把要操作对象的窗口激活后才能继续操作控件；
- Control 操作：窗口上的按钮、文本框等就是 Control 控件，前面提到了，要先激活窗口，再操作控件；
- Mouse 操作：AutoItLibrary 的鼠标操作要用到真实坐标，这和 Selenium2Library 里的坐标略有差异；
- Process 操作：这是对进程操作的，其实用 Robot Framework 自带的 Process 库就可以了；
- Run 操作：可以启动一个程序或者运行一个命令，同样用 Robot Framework 自带的 Process 库就可以了；
- Reg 操作：主要是对注册表进行操作。

2. 对象识别

在 AutoItLibrary 安装成功后，在硬盘的某个盘根目录会多一个 Robot Framework 的目录，具体在哪个盘取决于你的 User 目录在哪个盘。例如笔者的是在 D 盘，因此多出来的这个目录就在如下路径：

D:\RobotFramework\Extensions\AutoItLibrary

这里是一些辅助工具，比如 AutoItX.chm 是原生 AutoIt 的帮助文档，对于理解关键字的作用比较有帮助。AutoItLibrary.html 是测试库的关键字文档，它只是列出来所有的关键字和参数，基本很少有说明。Au3Info.exe 是最重要的识别对象的工具了。

打开 Au3Info.exe，在 Finder Tool 的位置有个“十字星”，可以用鼠标拖曳它到你需要识别的对象上。比如拖曳“十字星”到计算器的数字 4 上，会看到这样的界面，如图 4-3-1 所示。

再看图 4-3-1 右边的“AutoInfo”界面，可以看到识别出来最重要的属性分两块（在上半部分），即 Basic Window Info 和 Basic Control Info。

经过笔者多次使用，Window 方面识别用 Title 比较多，Control 主要用 controlID，controlID 就是在 Basic Control Info 里的 Class+Instance。比如图 4-3-1 中这个对象，它的 controlID 就是 Button4，关键字里的 strControl 就是 controlID（chm 里都是写的 controlID）。

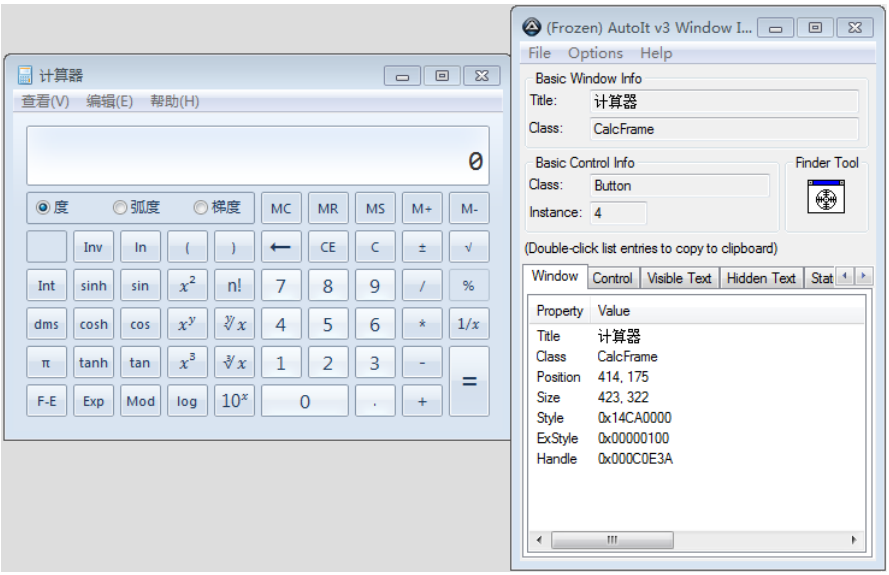

图 4-3-1

4.4 测试案例

4.4.1 计算器案例

在前面下载的 AutoItLibrary 安装包里有一个 tests 目录，tests 目录里面有一个对 Windows 的计算器做的测试案例，不过这个案例要稍微修改才能用，因为它是针对英文版 Windows 操作系统做的案例。英文版的计算器 Title 是 Calculator，而中文版的 Title 是计算器，所以要把案例中的 Calculator 修改成计算器才能使用。

在笔者的 demo 网站里的 AutoItDemo 的测试案例是笔者已经修改过的，适用于中文版 Windows 操作系统的案例。因为官方案例有很多，这里只展示一个修改过的案例。

首先，这里没有看到打开计算器的操作，实际上它放在用户关键字里了，然后在 Calculator Test Cases 这个 Suite 里面的 Setting 里有执行相应的关键字。这种处理方法推荐

大家学习一下。案例里面的 Click Buttons 是用户关键字，用来操作在计算器上输入后面的几个按键；Win Wait 是原生关键字，等待计算器这个窗口的出现；Get Answer 也是用户关键字，用了一个很巧妙的方法取到计算器上的值。具体方法这里不说，大家可以自己看案例，反正它不是用获取某个对象的属性值这种方式。拿到了实际值就可以做验证了，我们当然知道“41+1=42”，那么从计算器里获取的值是不是 42？就是 Should Be Equal As Numbers 这个关键字来做的验证了，这个关键字是 BuildIn 这个库里的关键字。脚本见表 4-4-1。

表 4-4-1

Click Buttons	4 1 + 1 =	
Win Wait	计算器	
${Ans} =	Get Answer	
Should Be Equal As Numbers	${Ans}	42

4.4.2　结合 Selenium2Library 处理对话框

AutoItDemo 下的 tests 目录里，第 1 个 Suite 是 Testsuite-Alert。这个 Suite 里主要是展示 Selenium2Library 和 AutoItLibrary 如何来处理 Web 上的对话框。

（1）Msgbox。你肯定见过这种对话框，有的是只有“确定”按钮的，这里是“确定”和“取消”两个按钮，如图 4-4-1 所示。

（2）InputBox。应该也见过这种对话框，如图 4-4-2 所示。

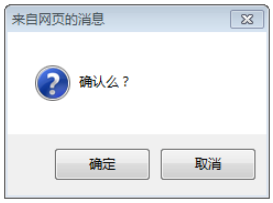

图 4-4-1

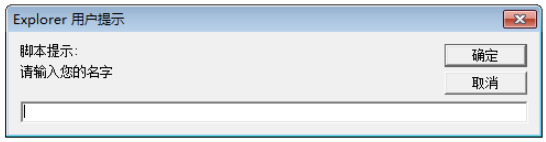

图 4-4-2

（3）用户密码认证框。还有那种在网页上弹出的要输入用户密码的登录框，主要是出现在用户认证不通过，返回 401 时，如图 4-4-3 所示。

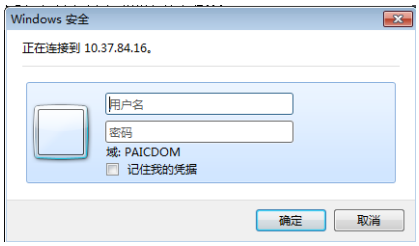

图 4-4-3

以上这些都可以用 AutoItLibrary 来处理。

对于第 1 种情况，Selenium2Library 中的 Confirm Action 可以处理，前面已经有这样的例子了。这里会用 AutoItLibrary 来实现同样的效果。

对于第 2 种和第 3 种情况，要用 AutoItLibrary 处理更好一些，因为那些文本框的输入已经脱离了 Selenium2Library 的控制了。

Tips： 本章的实际案例中还加了一些 Sleep，这里为了不占地方先去掉了，在笔者的 github 上的案例里是有的。本章后续的案例也是一样的。

1. 第 1 个案例是使用 Selenium2Library 的 Confirm Action 的案例

单击“pay”按钮时只有“OK”按钮的对话框，单击“paycon”按钮有“OK”和“Cancel”按钮的对话框，所以这里用到了两个关键字 Choose Cancel On Next Confirmation 和 Choose Ok On Next Confirmation。之前有很多人会把这两个关键字当作单击“OK”或者“Cancel”，实际上从字面意思上很清楚地说明了它是下一次 Confirm Action 时才会选择单击击“OK”或者“Cancel”，真正去单击的是“Confirm Action”，而不是“Choose”。脚本见表 4-4-2。

表 4-4-2

Click Button	pay	
${mes}	Confirm Action	
Choose Cancel On Next Confirmation		
click button	paycon	
${mes}	Confirm Action	
${confirm}	Get Element Attribute	r@innerText
Choose Ok On Next Confirmation		
Click Button	paycon	
${mes}	Confirm Action	
${confirm}	Get Element Attribute	r@innerText

2. 第 2 个案例是用 AutoItLibrary 来处理对话框

笔者只做了对 a 这种对话框的处理，b 和 c 的没做，其实只要学会了 a 这种如何用 AutoItLibrary 来处理，b 和 c 的就一样操作即可。

案例里的关闭对话框是用户关键字，笔者这里只做了单击“OK”的，如果要做单击“Cancel”的，大家可以自己修改一下。脚本见表 4-4-3。

表 4-4-3

Click Button	pay	
${mes}	关闭对话框	来自网页的消息
Click Button	paycon	
${mes}	关闭对话框	来自网页的消息
${confirm}	Get Element Attribute	r@innerText

关闭对话框的关键字内容如下，首先是根据 Title 判断这个窗口是否存在，然后使用 Control Get Text 获取对话框上的文本信息。前两个参数笔者用${EMPTY}表示传空值，第 3 个参数是 Static2，这个用的就是前面的 ControlID，获取的方法在前面也有介绍。然后打印获取到的文本${ret}，最后就是 Control CLICK 来单击“Button1”，脚本见表 4-4-4。

表 4-4-4

Win Wait	${title}	${text}	30	
${ret}=	Control Get Text	${EMPTY}	${EMPTY}	Static2
Log	${ret}			
Control CLICK	${EMPTY}	${EMPTY}	Button1	

4.4.3　结合 Selenium2Library 处理上传下载

Selenium2library 在实际测试 Web 页面时，基本上已经够用了。不过还是会在部分情况下会脱离 Selenium2library 的控制，无法进行操作。比如下载文件时，要选择保存文件在什么地方；比如上传文件时，要选择上传哪个文件。这些在 Selenium2library 下没有很好的处理办法，但是结合 AutoItLibrary 可以很好地进行处理。

1. 上传测试

（1）标准控件上传。

例如下面这种上传控件，如图 4-4-4 所示。如果是人工操作，用鼠标单击“浏览”按

钮来选择文件即可。

图 4-4-4

在 Selenium2Library 里面提供了处理这种控件的关键字，那就是 choose file。案例脚本见表 4-4-5。

表 4-4-5

Choose File	file	${CURDIR}${/}text.rar
${file}	Get Value	file

用这个关键字，会直接把文件路径加载到“浏览”前面的文本框里。

这里只是例子，正常应该另外会有个上传按钮，去单击进行真正的上传（以下皆同）。

（2）非标准控件上传。

例如下面这种只有一个按钮的，如图 4-4-5 所示。以前 CSDN 博客上传图片的选择文件按钮也是这种。

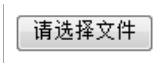

图 4-4-5

只能单击此按钮，在弹出的对话框里选择文件，这时就可以用上 AutoItLibrary 了。

由于笔者的 Demo 页面的这个按钮的例子是在网上找的，在具体实现用 Click Button 来单击按钮时，发现怎么都没法弹出来“选择文件”的对话框，无奈之下用上了一个“笨方法”来实现单击按钮，代码比较多，不过有效果。这个方法对于那种 flash 按钮的单击也可能有效。

说一下这个“笨方法”单击按钮的思路：

1）先通过 Selenium2Library 获取对象的坐标；

2）然后通过 AutoItLibrary 获取 IE 页面的坐标；

3）再通过 AutoItLibrary 获取 IE 窗口的真实坐标；

4）通过计算得到要单击按钮的真实坐标；

5）通过 AutoItLibrary 单击指定坐标的鼠标左键；

6）弹出了选择加载文件的对话框之后，可以直接在“文件名”的文本框中输入文件路径；

7）最后单击“确认”按钮。

案例脚本如下，2~3 行实现的是第 1 步的获取对象的坐标；4~5 行实现的是第 2 步的获取 IE 页面的坐标；6~7 行实现的是获取 IE 窗口的真实坐标；8~9 行实现的是第 4 步的通过计算得到真实坐标；剩下行的就是其余步骤的具体操作实现了。脚本见表 4-4-6。

表 4-4-6

${TITLE}	Get Title			
${cx}	Get Horizontal Position	choose		
${cy}	Get Vertical Position	choose		
${conx}	control Get Pos X			Internet Explorer_Server1
${cony}	control Get Pos Y			Internet Explorer_Server1
${ix}	win Get Pos X	${TITLE}		
${iy}	win Get Pos Y	${TITLE}		
${mx}	Evaluate	int(${cx})+int(${ix})+int(${conx})+18		
${my}	Evaluate	int(${cy})+int(${iy})+int(${cony})+38		
Mouse Click	LEFT	${mx}	${my}	
Win Wait	选择要加载的文件		20	
Win Activate	选择要加载的文件			
Control Set Text			Edit1	${CURDIR}${/}text.rar
Control Click			Button1	
${filepath}	Get Element Attribute	r@innerText		

2. 下载测试

单击页面的下载测试，会弹出“文件下载”提示框，想必大家也经常遇到，一般手动操作都是单击“保存”按钮，弹出“另存为”的对话框，如图 4-4-6 所示。先看看手动操作的完整步骤。

（1）单击页面下载测试链接，弹出页面后单击“保存”按钮。

（2）弹出“另存为”对话框，可以自己选择目录，或者在“文件名”文本框中输入路径来保存。

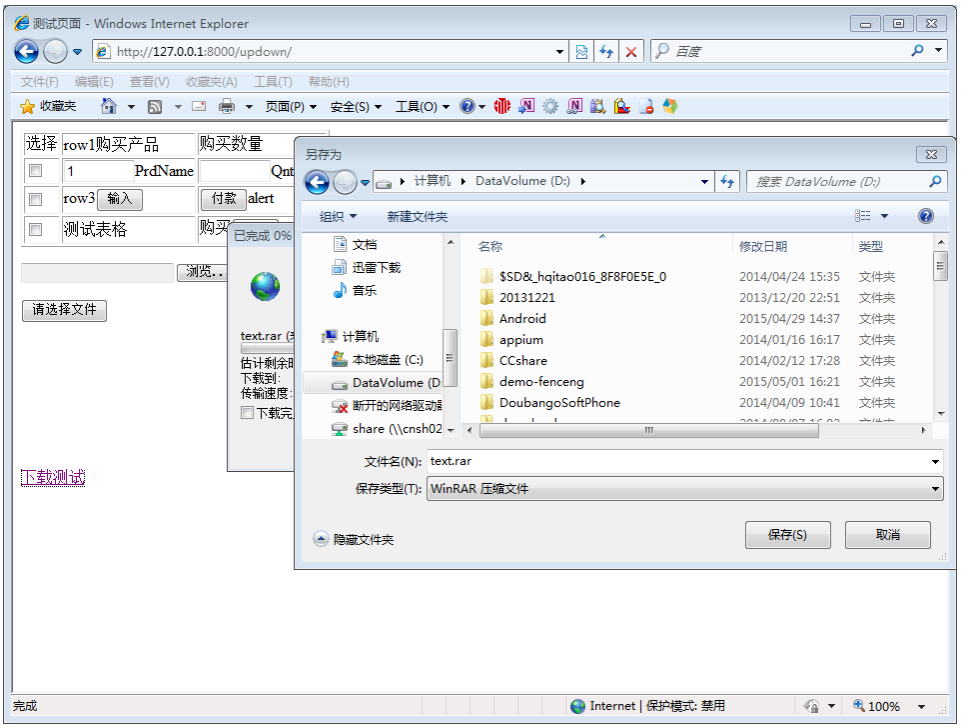

图 4-4-6

因为笔者做的是一个固定了路径的，每次下载都是到 Demo 目录下的同名文件，因此会出现存在同名文件的确认。

如果不想出现这个，就要想办法在保存文件的命名上，使文件名每次都不同，否则就要考虑这种流程的处理。

（3）保存同名文件的另存为处理。

（4）关闭下载完成的窗口。

操作完成后下面就是具体实现的案例脚本了。

1~4 行实现的是第 1 步的操作；5~8 行实现的是第 2 步的操作；9~11 行实现的是第 3

行的操作；12~14 行实现的是第 4 步的操作。脚本见表 4-4-7。

表 4-4-7

Click Link	下载测试			
Win Wait	文件下载		15	
Win Activate	文件下载			
Control Click			Button2	
Win Wait	另存为		15	
Win Activate	另存为			
Control Set Text			Edit1	${CURDIR}${/}text2.rar
Control Click			Button1	
${confirm}	Win Exists	确认另存为		
Run Keyword If	${confirm}==1	Win Activate	确认另存为	
Run Keyword If	${confirm}==1	Control Click		Button1
Win Wait	下载完毕			
Win Activate	下载完毕			
Win Close	下载完毕			

4.5　小结

AutoItLibrary 本身是可以作为 C/S 架构的程序自动化测试来使用的，官方的计算器的例子就是这样的。只不过它只适用于标准 Windows 32 位操作系统的控件，非标准 Windows 32 位操作系统的控件基本上不适合用 AutoItLibrary。如何知道是否是标准 Windows 32 位操作系统的控件，就用 Au3Info.exe 去识别一下对象。只要 control 下面的 class 出现是正常的英文名，都可以来识别，比如 Edit/Button 之类的。

5

第 5 章 数据库自动化测试

本章主要介绍如何对数据库进行自动化测试，其实更多的是进行数据构造。这里会介绍 Oracle 和 sqlite3 的自动化案例。

本章的测试案例：

https://github.com/qitaos/rf-demos/tree/master/DatabaseDemo

5.1 数据库测试介绍

在测试过程中不可避免的会用到数据库，无论是测试数据还是查询数据，都需要我们用到数据库，同时有时候也需要对数据库进行测试。常见的数据库有 Oracle、MySQL、sqlite3 等。每个数据库都需要有相应的连接方法，幸好有 DatabaseLibrary，它能够兼容多种数据库，把这些数据库统一连接起来，不过有些数据库仍然需要相应的工具补充支持。

5.2　DatabaseLibrary 和 cx_Oracle 安装

DatabaseLibrary 的安装，官方下载地址：

https://github.com/franz-see/Robotframework- Database-Library

除了这个测试库外，如果想对 Oracle 进行测试，建议安装 cx_Oracle，下载地址：

http://sourceforge.net/projects/cx-oracle/files/

选择 cx_Oracle 安装包的时候要注意选择正确的安装包，首先看你本地的 Oracle 客户端是 10g 还是 11g，其次是 Windows 操作系统是 32 位的还是 64 位的，然后选择支持 py27 的安装包（本书中的 Oracle 都是在 Windows 操作系统下的）。

5.3　DatabaseLibrary 关键字

DatabaseLibrary 的关键字不多，有以下几个。

- connect to database：连接数据库，标准连接；
- connect to database using custom params：使用自定义参数连接数据库；
- disconnect from database：（使用完数据库后记得）断开数据库；
- query：用于查询数据库，可以用 List 变量接收返回结果；
- row count：用于查询 sql 语句的结果条数，其实用 query 也可以；
- execute sql script：用于执行 sql 文件，可以把 sql 脚本保存到 sql 文件里执行，注意脚本要用英文分号“;”分隔开，最后一个脚本不要带英文分号“;”；
- execute sql string：用于执行 sql 脚本，和上一个关键字差不多，只不过脚本是直接写出来，不是在文件里。注意脚本要用英文分号“;”分隔开，最后一个脚本不要带英文分号“;”。

其他还有几个断言的关键字，没有列出来。

5.4　测试案例

5.4.1　Oracle 数据库

这个例子笔者会给出脚本和执行结果，但是由于没法单独弄个 Oracle 数据库给 Demo

案例用，提供 Demo 给大家也用不了，不过笔者录制了相关的视频。下面的 sqlite3 的 Demo 是可以执行的。

1. 案例 1

这里连接数据库用的是 Connect To Database Using Custom Params，因为连接 Oracle，所以第 1 个参数用的是 cx_Oracle，第 2 个参数里的内容分别是用户名、密码和数据库标识 sid，数据库标识需要你在 TNSNAMES.ORA 里配置好相应的连接串。

Row Count 用来查询后面的 Sql 执行结果的行数，你也可以自己用 Query 来执行 Sql 自带 count 的，不过结果要再处理一下才能用。

Execute Sql String 可以用来执行多个 sql 脚本，如果是普通的 sql 脚本，比如 insert/update/delete 之类的，可以一起执行。但是要注意一个规则，就是它们要用分号“;”分隔开，最后一个脚本不用加“;”。如果只有一条语句也不用加“;”，这个规则基本上在 DatabaseLibrary 里是通用的，在 Execute Sql String 的关键字说明里有例子。但是有一个特例，就是执行存储过程，所以这里的例子就用它了，可以看到脚本最后用了“end;”，它是带“;”的，如果不带就会报错，这一点请注意。脚本见表 5-4-1

表 5-4-1

Connect To Database Using Custom Params	cx_Oracle	'pub_test', 'pub_test', 'd0cfs'
${row}	Row Count	select * from pubtest_pa18infor_mas
@{rs}	Execute Sql String	begin pub_test.pkg_test_cgi.proc_cgi_proRpyPlan(); end;
Log many	@{rs}	
${row}	Row Count	select * from pubtest_pa18infor_mas
Disconnect From Database		

2. 案例 2

这里大家注意到笔者换了一种写法，前一种是用数据库标识 sid 连接，如果你没有配置的话，也可以用 DESCRIPTION 来连接，当然这种方法写起来比较长。两种方法都可以，推荐前面一种。

得到 Query 的值之后如何取值呢？以 gen_num 为例，获取到的值应该是[('123456',)]这样的，“[]”是整个的结果集 List，每一个“()”都是一条记录，“()”里面用逗号分隔开的是数据的列。在 Python 里“[]”是 List，“()”是 Tuple 元组，两者确实有些差异，不过在这里，你可以把这个结果集理解为一个二维数组。前面的章节介绍过了，二维数组要

用${num[0][0]}这样的写法来取值，不能用@{rs}[0][0]来取值，这里的案例也是重新给大家复习一下这个知识点。脚本见表 5-4-2。

表 5-4-2

Connect To Database Using Custom Params	cx_Oracle	'pub_test', 'pub_test', '(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=d0cfs.dbdev.paic.com.cn)(Port=1526))(CONNECT_DATA=(SID=d0cfs)))'
@{rs}	Query	select * from pubtest_pa18infor_mas
Log many	@{rs}	Log many
Log	@{rs}[0][0]	
${num}	Query	select gen_num(6) from dual
Log	${num[0][0]}	
Disconnect From Database		

5.4.2　sqlite3 数据库

默认安装 python 的时候就装了 sqlite3 数据库，这个数据库很轻量，也很方便，用来做例子演示再好不过了。

1. 案例 1

案例 1 里做了很多操作，连接 sqlite3 数据库比较简单，直接把文件路径用单引号引起来，放在第 2 个参数就行了。

大多数关键字前面都介绍过了，第 2 行是查询一下 sqlite_master，显示一下当前表的索引和表结构相关信息。4~9 行主要是查询当前表中最大的 id，然后加 1 之后 insert 一条新记录进去，然后查询一下数据，打印出最后一条记录${ao[-1]}。10~11 行是两个判断的关键字，第 1 个判断不存在记录，第 2 个判断存在记录。由于两个判断暂时都是肯定会失败的，所以加了一个 Run Keyword And Ignore Error 来忽略错误。脚本见表 5-4-3。

表 5-4-3

Connect To Database Using Custom Params	sqlite3	'${CURDIR}${/}demo.db'	
${table}	Query	select * from sqlite_master where type = 'table'	

续表

${a}	Row Count	select * from order_item where name='范德萨防守打法撒'	
${id}	Query	select max(id)+1 from order_item	
${id}	Evaluate	'${id}'.replace('None','1')	
${id}	Evaluate	${id}	
${d}	Execute Sql String	insert into order_item values (${id[0][0]},'product${id[0][0]}', ${id[0][0]})	
${ao}	Query	select * from order_item	
Log Many	${ao[-1]}		
${check1}	Run Keyword And Ignore Error	Check If Not Exists In Database	select * from order_item where id=1
${check2}	Run Keyword And Ignore Error	Check If Exists In Database	select * from order_item where id=10000
Disconnect From Database			

2. 案例 2

这里没什么特别介绍了，主要是查询结果的值，如何获取的方式。脚本见表 5-4-4。

表 5-4-4

Connect To Database Using Custom Params	sqlite3	'${CURDIR}${/}demo.db'
${id}	Query	select * from order_item
Log	${id[0][1]}	
Disconnect From Database		

5.5 常见问题

1. Dll load failed

如果在使用中出现“Dll load failed”，则需要到 Oracle 网站下载 instantclient 的压缩包，然后将其中的 oraociei11.dll、oraocci11.dll、oci.dll 这 3 个文件复制到 python 安装目录下。

2. Windows 64 位操作系统上的错误

笔者之前遇到的问题是在 64 位机器上使用 cx_Oracle，经常出现各种报错。主要原因是笔者的 python 安装的是 32 位的，对应的 cx_Oracle 和 Oracle 客户端也都是 32 位的。当时作为疑难问题“挂了”很久，后来把它们全都换成 64 位的就解决了。

3. 数据库中文处理

经常会有人反馈，数据库查询结果返回的内容的中文显示不正确。如果你看到的中文是“\u”开头或者“\x”开头的，“\u”的其实没什么问题，直接使用就行了；“\x”的要用 Evaluate 处理成中文。下面是一个处理“\x”开头的例子，如果是手写要用“\\x”，如果是变量就直接放到单引号里。脚本见表 5-4-5。关于 Evaluate 的用法，后面第 7 章会有详细介绍。

表 5-4-5

${text}	Evaluate	'\\x3ad\\x333'.decode('gbk')

如果看到的是“???”这样的文本，有两种处理方式。一种是在 Windows 的环境变量里添加一个变量，名为 NLS_LANG，值为 SIMPLIFIED_CHINESE_CHINA_UTF8；另一种和它一样，只不过用 RF 的方式来做，用下面的脚本，见表 5-4-6。

表 5-4-6

Set Environment Variable	NLS_LANG	SIMPLIFIED_CHINESE_CHINA_UTF8

这个关键字在 RF 的内置测试库 OperatingSystem 里，所以要加载测试库 OperatingSystem。

5.6　小结

其实专项进行数据库测试的并不太多，根据个人的经验，实际上更多的是结合前台的流程，使用数据库测试方法来进行数据准备或数据验证。大家也可以根据自己项目的需要，酌情使用数据库测试的方法。这里只介绍了 Oracle 和 sqlite3 的数据库，实际上 DatabaseLibrary 是支持很多数据库的，只不过连接数据库的方式可能多少会不同。由于笔者这里没有环境，所以就没有提供更多的数据库的例子了。如果以后有例子了，笔者会补充到 Github 上。

6

第 6 章
接口自动化测试

这一章主要介绍几种常见的接口如何进行自动化测试，主要介绍了 get request 和 post request。

本章的测试案例：

https://github.com/qitaos/rf-demos/tree/master/RequestsDemo

6.1 接口测试

接口测试通常是系统之间交互的接口，或者某个系统对外提供的一些接口服务，笔者之前接触的大部分都是 RESTful，还有一些 Web Service 的接口。现在大家也越来越关注接口测试了，因为有时候可能界面上的功能还没有实现，可以先对接口进行验证，提早发现一些和预期不一致的错误。这方面的工具应该也有一些，基于 RF 的接口测试库，这里

首推 requests 和 requestsLibrary。

6.2　requestsLibrary、requests 安装

requests 的官方下载地址：

https://pypi.python.org/pypi/requests

requestsLibrary 的官方下载地址：

https://pypi.python.org/pypi/robotframework-requests/

requestLibrary 就是用于 Robot Framework 的测试库，底层基于 requests 这个工具。安装和之前一样，下载 tar.gz 包安装，先要安装 requests，再安装 requestsLibrary。

6.3　requestsLibrary 关键字

requestsLibrary 的关键字不多，常用的有以下几个。

- Create Session：创建一个 session，连接某个服务器；
- Create Ntlm Session：也是创建一个 session，只不过加上了域、用户名、密码用于 NTLM 认证；
- Get Request：通过 GET 方式发起请求；
- Post Request：通过 POST 方式发起请求；
- Head Request：发送一个 HEAD 请求；
- TO Json：将文本转换成 json 对象。

6.4　测试案例

这一章的 Demo 也是用到了 flask 的网站，做了几种例子，大家了解一下。如果需要更完整的使用，建议看一下 RequestsLibrary 在 github 上的测试案例。

1. get request json

访问 http://localhost:8000/users/1 的时候，它会返回一个 json：

```
{"1": "john"}
```

如果把 1 换成 5，它会返回一个 404 的 json：

```
{
"message": "Not Found: http://localhost:8000/users/5",
"status": 404
}
```

所以这里的案例就是来测试这 2 个接口请求的。

首先要先 Create Session 创建一个连接到服务器的 host，然后通过 Get Request 发起请求，请求的返回是个 Request 对象${addr}。所以在这里的变量是个对象，而对象的属性值有${addr.status_code}和${addr.content}，使用了 To Json 后，就把返回的 content 格式化成 Json 串，然后就可以用 Dictionary 的方法获取其中的值了。脚本见表 6-4-1。

表 6-4-1

Create Session	api	http://localhost:8000	
${addr}	Get Request	api	users/1
Should Be Equal As Strings	${addr.status_code}	200	
Log	${addr.content}		
${responsedata}	To Json	${addr.content}	
${keys}	Get Dictionary Keys	${responsedata}	
${items}	Get Dictionary Items	${responsedata}	
${values}	Get Dictionary Values	${responsedata}	
${str}	Get From Dictionary	${responsedata}	1
${addr}	Get Request	api	users/5
Should Be Equal As Strings	${addr.status_code}	404	
Log	${addr.content}		
${responsedata}	To Json	${addr.content}	
${keys}	Get Dictionary Keys	${responsedata}	
${items}	Get Dictionary Items	${responsedata}	
${values}	Get Dictionary Values	${responsedata}	
${str}	Get From Dictionary	${responsedata}	message
Delete All Sessions			

2. get request xml

前面一个例子是接口返回 json 进行处理的，这个例子是接口返回 xml 格式内容。访问

http://localhost:8000/hello/qitao，会返回一个 xml 格式的内容：

```
<response>
<hello>qitao</hello>
</response>
```

案例比较长，分两个部分说。第一个部分其实还是用 json 来处理，虽然返回的是 xml，但是 request 自动把它处理成 json 了。所以它的这部分处理其实和前面的案例一样。脚本见表 6-4-2。

表 6-4-2

Create Session	api	http://localhost:8000	
${addr}	Get Request	api	hello/qitao
Log	${addr.content}		
${responsedata}	To Json	${addr.content}	
${keys}	Get Dictionary Keys	${responsedata}	
${items}	Get Dictionary Items	${responsedata}	
${values}	Get Dictionary Values	${responsedata}	
${str}	Get From Dictionary	${responsedata}	hello

那么想要返回 xml 怎么处理呢？这里就要在 Request 的 header 里加上一个 accept=application/xml，此时返回的就是一个 xml 格式的内容：<?xml version="1.0" ?><response><hello>qitao</hello></response>。

所以这里用到了 XML 库的关键字 Get Element Text 来获取 xml 节点的文本，使用 Get Element 获取到一个 Element 对象${hello}。既然是对象，那么它也有属性值${hello.text}。再后面的几行就是用 XML 库的关键字新增 Element，然后取值，这也算是提前准备好来 XML 的 Demo 了。

这里为什么要做 xml 的例子呢？其实是因为有一种接口测试就是测试 Web Service 的，通常 Web Service 都是返回一个 xml 格式的内容。这里 Demo 网站没有做 Web Service 的例子，所以就用这个 xml 的例子来代替 Web Service 了。脚本见表 6-4-3。

表 6-4-3

${dict}	Create Dictionary	accept=application/xml		
${addr}	Get Request	api	hello/qitao	${dict}
Log	${addr.content}			
${responsedata}	Set Variable	${addr.content}		

续表

${body}	Get Element Text	${responsedata}	hello	
${hello}	Get Element	${responsedata}	hello	
Log	${hello.text}			
${responsedata}	Add Element	${responsedata}	<new id="3">test</new>	
${new}	Get Element Attribute	${responsedata}	id	new
Log	${new}			
${a}	Element To String	${responsedata}		
Delete All Sessions				

3. 增加登录态

有些接口是需要用户用密码登录后才能访问的，如果直接访问 http://localhost:8000/401，会提示请输入用户名密码，输入正确的用户名密码（ok/python），会得到一个返回的信息：{"pass": "Authorized access"}。

此时可以在 Create Session 的时候加上用户名密码，就可以直接访问了。

用户名是“ok”，密码是“python”，用${auth}创建个 List 存起来，然后在 Create Session 时加上${auth}参数，后面的过程就和前面一样了，不需要自己再手动输入用户密码了。脚本见表 6-4-4。

表 6-4-4

${auth}	Create List	ok	python	
Create Session	api	http://localhost:8000		${auth}
${addr}	Get Request	api	401	
Log	${addr.content}			
${responsedata}	To Json	${addr.content}		
${keys}	Get Dictionary Keys	${responsedata}		
${items}	Get Dictionary Items	${responsedata}		
${values}	Get Dictionary Values	${responsedata}		
${str}	Get From Dictionary	${responsedata}	pass	
Delete All Sessions				

4. post request

前面几个都是 get 方式的，这个例子是 post 方式的。

这里主要是 header 加了一项 Content-Type=application/x-www-form-urlencoded，这是最常见的 post 提交数据的方式。另外还有几种，大家可以根据自己的实际情况来使用。

然后向 http://localhost:8000/post 发起请求，并将${data}作为 post 的 data 传过去。Demo 网站对 post 请求做了个处理，将 username 的值获取到，然后以 json 格式返回。脚本见表 6-4-5。

表 6-4-5

${dict}	Create Dictionary	Content-Type=application/x-www-form-urlencoded		
Create Session	api	http://localhost:8000	${dict}	
${data}	Create Dictionary	username=qitao	password=qt	
${addr}	Post Request	api	post	data=${data}
Log	${addr.content}			
Log	${addr.json()}			
${responsedata}	To Json	${addr.content}		
${keys}	Get Dictionary Keys	${responsedata}		
${items}	Get Dictionary Items	${responsedata}		
${values}	Get Dictionary Values	${responsedata}		
${str}	Get From Dictionary	${responsedata}	username	
Delete All Sessions				

6.5　小结

接口测试其实比前面的 UI 自动化测试要有价值得多，而且投入也不是很高，维护成本也低很多，一般来说接口上不会特别频繁地变更。即使有变更，正好也可以用这些案例来回归其他接口是否受到了影响。所以推荐大家将接口自动化测试优先做起来，一定要在上线前跑接口自动化回归测试，某些时候能够避免一些接口变更导致的问题。

7

第 7 章

RF 内置测试库

Robot Framework 框架自身内置了一些常用的测试库，这里会介绍几个主要的测试库的关键字以及 demo 测试案例。

本章的测试案例：

https://github.com/qitaos/rf-demos/tree/master/RF-Libraries-Demo

7.1 测试库介绍

Robot Framework 本身自带了一些常用的测试库，它们的位置是在“Python 目录”\Lib\site-packages\robot\libraries。

另外在 RIDE 里有个小问题，就是 RIDE 里加载这些内置库的时候，它显示的关键字其实是从 RIDE 目录里面的 lib\robot\libraries 目录下读取的。有时 Robot Framework 出了新版，RIDE 里没有及时更新，可能就会导致你使用新的关键字不被 RIDE 识别成关键字。

此时可以做一件事情，就是把“Python 目录”\Lib\site-packages\robot\libraries 里的测试库同步复制到“Python 目录”\Lib\site-packages\robotide\lib\robot\libraries 目录下，这样在 RIDE 里看到的内置测试库的关键字就都是最新的了。

如果想查看这些内置测试库的文档可以访问如下地址：

http://robotframework.org/robotframework/

这里有完整的测试库文档（英文的）。很多人看到英文文档就害怕，其实笔者英文也不是很好，但是在学习 Robot Framework 的过程中都是自己去看官方文档的，如果遇到不会的就自行翻译。所以笔者并不希望把后面例子的内容做成一个关键字列表，因为这等于去翻译了一遍官方的文档了。笔者希望的是通过这些例子，让大家明白如何使用测试库的关键字。所以会用其中常用的几个测试库来举例，例子里不一定会有该测试库的全部关键字，大家看了例子，只要明白了用法就可以融会贯通了。

Tips：其实不管哪个测试库，强烈建议大家详细地看一下测试库的文档，或者把所有的关键字都看一遍，知道了关键字有哪些用途，这样才有助于在后续设计测试案例时使用。

7.2　BuiltIn

BuiltIn 这个测试库从来没有被我们手动添加过，但是按 F5 快捷键的时候，它就在那里，已经被加载了。这个主要是在 RIDE 的源码中默认初始化时就会加载 BuiltIn 库。

BuiltIn 里面提供的关键字很多，笔者会按照它自身的大类分开，介绍每类中常见的关键字用法，有很多类似的关键字没有一一列举。

7.2.1　Convert

这一部分都是转换用的关键字，做了 2 个案例的 demo。

第一个案例是默认把十进制的数字，转换成其他进制或者别的值。下面这些都是常见的转换，其中 Number 就是 python 里的 float 值。脚本见表 7-2-1。

表 7-2-1

${a}	Set Variable	100
${integer}	Convert To Integer	${a}

续表

${number}	Convert To Number	${a}
${binary}	Convert To Binary	${a}
${octal}	Convert To Octal	${a}
${hex}	Convert To Hex	${a}
${string}	Convert To String	${a}
${bytes}	Convert To Bytes	${a}
${boolean}	Convert To Boolean	${a}

第二个案例是在第一个案例的基础上加了些参数，前面都是${a}默认为十进制，这里可以提供参数说明当前是哪个进制。比如可以从八进制转换为二进制，详细的转换示例大家可以查看一下关键字说明。脚本见表 7-2-2。

表 7-2-2

${a}	Set Variable	100	
${integer}	Convert To Integer	${a}	2
${binary}	Convert To Binary	${a}	8
${octal}	Convert To Octal	${a}	16
${hex}	Convert To Hex	${a}	10
${string}	Convert To String	${a}	
${bytes}	Convert To Bytes	${a}	
${boolean}	Convert To Boolean	${a}	

7.2.2 Verify

这个部分做了 4 个案例的 demo。

第一个案例，这里只是介绍一下 Fail，你可以自己增加某些条件的判断来执行 Fail。当 Fail 执行后，当前的案例会停止，后续的案例会继续执行。脚本见表 7-2-3。

表 7-2-3

Fail	停止当前Case
Log	不会打印

第二个案例，Fatal Error 和 Fail 的差别就是它会停止所有的案例，包括在当前案例之后没有运行的案例。脚本见表 7-2-4。

表 7-2-4

Fatal Error	停止所有Case
Log	后面的案例都会停止

第三个案例，这里有很多 Should 的断言，可以根据自己的需要来选择使用。实际的例子里脚本很多，这里贴出一部分以供参考，见表 7-2-5。

表 7-2-5

Should Not Be Empty	A	
Should Be Empty	${EMPTY}	
Should Not Be True	0	
Should Be Equal	2	2
Should Start With	HELLO	HE
Should End With	HELLO	LO
Should Match	AAAB	A*B
Should Match Regexp	2	\\d

第四个案例主要就是其他的几个关键字。Get Count 是获取第二个参数在第一个参数里有几个。Get Length 就是获取 List 变量里的元素个数，如果后面的参数是 Scalar 变量，就是获取字符的个数了。脚本见表 7-2-6。

表 7-2-6

${count}	Get Count	hello world	o
@{list}	Create List	1	2
${len}	Get Length	${list}	
Length Should Be	${list}	${len}	

7.2.3　Variables

这个部分主要是一些和变量有关的关键字。

变量是有作用域的，所以这边有 Set Test、Suite、Global Variable 来分别给不同的作用域设置变量。Get Variable Value 可以用来获取变量的值，它有 2 个参数，如果第一个参数的变量没有找到，那么就获取第二个参数里变量的值，默认第二个参数的值是 None。脚本见表 7-2-7。

表 7-2-7

${var}	Set Variable	a	
${var2}	Set Variable If	'${var}'<> '0'	9
Set Test Variable	${testvar}	test	
Set Suite Variable	${suitevar}	suite	
Set Global Variable	${globalvar}	global	
${getvar}	Get Variables		
${varval}	Get Variable Value	${var}	
${varval}	Get Variable Value	${var3}	${var2}
Log Variables			
Variable Should Exist	${testvar}		
Variable Should Not Exist	${qtvar}		
${globalvar}	Replace Variables	${testvar}	

7.2.4 RunKeyword

这个部分是结合关键字使用的一些关键字，其中有部分关键字需要在 Teardown 里才能使用。例如 Run Keyword If Test Failed，这类关键字有几个，都是判断案例的运行结果后，用来执行其他关键字操作的。

demo 案例里主要是做了一些满足条件，然后用 log 来打印一些文字的操作，基本上只要理解了关键字的字面意思，就能看明白每一个步骤的含义。

需要说明的是，其中${var}是一个变量，但是把 log 作为它的值之后，它就可以在 Run Keyword 系列关键字里把变量的值当作关键字使用。第 4 行的第一列三个点是换行的意思，表示三个点后面的几个关键字，是跟着上一行关键字最后的参数连着的。脚本见表 7-2-8。

表 7-2-8

${var}	Set Variable	log	
Run Keyword	${var}	abc	
Run Keywords	${var}	abc	
...	AND	${var}	EFG
Run Keyword If	'${var}' == 'log'	${var}	iflog
Run Keyword Unless	'${var}' == 'test'	${var}	unless

续表

Run Keyword And Ignore Error	Fail	ignore	
${status}	Run Keyword And Return Status	kw1	selenium
Run Keyword And Continue On Failure	Fail		
Run Keyword And Expect Error	*	Fail	
Repeat Keyword	2	kw1	repeat
Wait Until Keyword Succeeds	30s	5s	kw1

7.2.5　Control

这个部分主要是一些与流程控制相关的关键字。

第一个案例里，Continue For Loop 是继续执行下一个循环，当前循环内后面的脚本不再执行。Exit For Loop If 是满足条件时退出循环。Pass Execution 就是直接给一个 Pass 的结果，当前案例后面的脚本都不会再执行了。脚本见表 7-2-9。

表 7-2-9

:FOR	${in}	IN RANGE	5
	Log	${in}	
	Run Keyword If	${in} == 3	Continue For Loop
	Log	cfloop=${in}	
:FOR	${in}	IN RANGE	5
	Log	${in}	
	Exit For Loop If	${in} == 2	
	Log	exitloopif=${in}	
Pass Execution	pass后面的不会执行		
Pass Execution If			

第二个案例主要是体现了 Return From Keyword 和 Run Keyword And Return 的用法，一般都是在用户关键字的 Return Value 里设置返回值。第一个关键字，实现了在程序中间进行返回，可以看出是提高了效率。第二个关键字，实际上实现了一个闭包的关键字处理，调用另一个关键字，然后把那个关键字的返回值作为当前关键字的返回值返回。

主要的脚本在关键字里，这里是用来找一个元素是否在 List 里存在。如果存在就返回它的${index}，如果不存在就返回-1。脚本见表 7-2-10。

表 7-2-10

${index} =	Set Variable	${0}		
: FOR	${item}	IN	@{items}	
	Run Keyword If	'${item}' == '${element}'	Return From Keyword	${index}
	${index} =	Set Variable	${index + 1}	
Run Keyword And Return	kw2	${-1}		

7.2.6 Misc

这一部分就是比较杂乱的了，有一些不太常用的去掉了，下面这些是响应的关键字用法示例。No Operation 通常用在还没设计好的用户关键字里，因为关键字里不写任何脚本执行案例会报错，所以加上这个关键字就不会有报错，也不会做任何操作。Catenate 是拼接字符串，可以指定分隔符是什么。其他的几个关键字都很常用，就不一一介绍了。脚本见表 7-2-11。

表 7-2-11

No Operation			
Sleep	2s		
${cate}	Catenate	hello	world
Comment	这是注释		
Set Log Level	trace		
Log	log文本		
Log Many	a	b	c
Log To Console	console		

7.2.7 强大的 Evaluate

有的时候，测试库也未必能满足我们的需要，这时候就需要有一个强大的关键字来帮我们——Evaluate。它可以做很多事情，主要的作用是可以直接调用 Python 的方法，此外对于使用中文的朋友在遇到一些乱码的时候，也可以用它转换成可以识别的中文。接下来笔者会分几个不同的常见来列举这个关键字的使用。实际上它的某些用法也有对应的关键字可以使用，但是一般都不在 BuildIn 里，要再额外加载一个 Library。

1. 随机数

一般用 Evaluate 都是前面放变量接收值，第三列是具体的运算表达式，第四列是要用

到的 Python 的 module。这里就是用 random 来进行一个随机数的生成。脚本见表 7-2-12。

表 7-2-12

${num}	Evaluate	random.randint(0,10000)	random

2. 随机字符串

首先是通过 string 来获取到一个全是大写的 ascii 英文字母全集，然后勇敢 len 获取它的字符个数，这里面第三行的 Get Length 其实和 len 是一样的，只是两种不同的写法，结果是一样的。写在这里是为了表明某些关键字能做到的事情，Evaluate 也能做到。之后就是设定${num}为 4，获取 4 个随机字符。然后通过 FOR 循环，每次随机从${str}里取一个字母，然后拼接起来放到${newstr}里，循环结束也就完成了随机字符的生成。

当然，这里主要是为了演示 Evaluate 的作用，后面在 String 库里，一个关键字就可以做到下面这么多脚本做的事情。脚本见表 7-2-13。

表 7-2-13

${str}	Evaluate	string.ascii_uppercase	string	
${strlen}	Evaluate	len('${str}')		
${len}	Get Length	${str}		
${num}	Set Variable	4		
${newstr}	Set Variable	${EMPTY}		
: FOR	${index}	IN RANGE	${num}	
	${i}	Evaluate	random.randint(0,int(${len})-1)+1	random
	${temp}	Set Variable	${str[int(${i})-1]}	
	${newstr}	Set Variable	${newstr}${temp}	
Log	${newstr}			

3. 字符串处理

首先给${str}赋值，两头加了 2 个空格的 hello world，这样是为了第三行的对它进行 strip()，去除两边的空格。然后第五行用 replace 来把字符串里的“o”换成“h”，第六行则是将空格替换成空，也就是去掉空格。最后第九行就是用了一下 string.join 来拼接字符串。脚本见表 7-2-14。

表 7-2-14

${str}	Set Variable	\ \ hello world \ \	
Log	=${str}=		
${str}	Evaluate	'${str}'.strip()	
Log	=${str}=		
${str}	Evaluate	'${str}'.replace('o','h')	
${str}	Evaluate	'${str}'.replace(' ','')	
${urlA}	Set Variable	http://www.baidu.com	
${urlB}	Set Variable	more	
${url}	Evaluate	string.join(['${urlA}','${urlB}'],'/')	string

4. 中文处理

中文处理是 python 里的一个老大难的问题，通常在测试案例中最常见到类似于“u'\\u4e2d\\u6587'”或者“\\xd6\\xd0\\xce\\xc4”这两种。前者是 unicode 的字符，后者是 gbk 的字符。我们的 RIDE 里默认的就是 unicode，所以如果见到前面这种，基本上直接用就行了。如果一定要看到中文，就用 Evaluate“过一道”，如同第二行那样。如果是后者，可以用第四行的方法来处理，就会变成中文了。后面的脚本是笔者试图用 Evaluate 来把中文先转码，然后再转回中文，不过只有 gbk 的“\\x”是可以的，UTF-8 的只有在 List 里才会显示出“\\u”的。大家只要记住前面两种处理方式就可以了。脚本见表 7-2-15。

表 7-2-15

${utf8}	Set Variable	u'\\u4e2d\\u6587'
${utf8cn}	Evaluate	${utf8}
${gbk}	Set Variable	\\xd6\\xd0\\xce\\xc4
${gbkcn}	Evaluate	'${gbk}'.decode('gbk')
${a}	Set Variable	中文
${utf8}	Evaluate	'${a}'.decode('utf-8')
${utf8cn}	Evaluate	u'${utf8}'
${utflist}	Create List	${a}
${gbk}	Evaluate	'${a}'.decode('utf-8').encode('gbk')
${gbkcn}	Evaluate	'${gbk}'.decode('gbk')

5. 正则表达式

正则表达式其实笔者不太熟悉，这里举几个简单的例子。

第二行的 re.subn 是将字符串里不是英文和数字的字符替换成下划线“_”，返回值是替换后的字符串和替换的次数。第四行的 re.sub 是把字符串里的数字都换成“*”。最后两个 findall 都是获取字符串里的数字，它们的差异只是一个是“\\d”，另一个是“\\d+”。“\\d”就是把数字一个一个取出来，“\\d+”也是把数字取出来，如果是连续的数字作为一个值取出。所以前者取出来的是['2', '5', '2', '3', '6', '9', '0']，后者取出来的是['2', '523', '6', '90']。脚本见表 7-2-16。

表 7-2-16

${rm}	Set Variable	paic-2.523.6-90	
${subn}	Evaluate	re.subn('[^\\w]','_','${rm}')	re
Log	${subn[0]}		
${sub}	Evaluate	re.sub('[\\d]','*','${rm}')	re
${findnumber}	Evaluate	re.findall('\\d','${rm}')	re
${findnumbers}	Evaluate	re.findall('\\d+','${rm}')	re

6. 日期处理

Get Time 其实已经可以很好地帮我们进行日期处理了，但是涉及时间加减，它只能加减处理和 NOW 或 UTC 有关的日期时间。如果有一个指定日期的字符串，想要进行时间加减的处理，Get Time 就不太好用了。像第一行这样用“NOW+1day”的获取日期是可以的，但是如果有一个指定日期，例如 2014-10-15，要用 Get Time 进行加减就比较麻烦，反正笔者试验了很久也没找到写法。这里在第三行分别取出年、月、日，然后在 Evaluate 里面对年、月、日重新处理成 datatime，然后用 datetime.timedelta 来进行时间的加减。此处设置的是减一天，处理之后用 split 把年、月、日分开。最后一行是 time 的日期时间字符串的处理。脚本见表 7-2-17。

表 7-2-17

${ymd}	Get Time	year month day	NOW-1day	
${gettime}	Get Time	year month day	2014-10-15	
${year}	${month}	${day}	Set Variable	${gettime}
${addDays}	Set Variable	-1		
${newDate}	Evaluate	datetime.date(int('${year}'),int('${month}'),int('${day}'))+datetime.timedelta(days=int('${addDays}'))	datetime	
${newYMD}	Evaluate	'${newDate}'.split('-')		
${newTime}	Evaluate	time.strftime("%Y-%m-%d-%H-%M-%S")	time	

7. 执行命令

Evaluate 也可以用来执行命令。这个案例是用 Mac 操作系统的电脑来写的案例，所以笔者加了个 sys.platform 的判断。之后用 mkdir 创建了目录 qttest，然后执行 ls，最后用 rm –rf 删掉 qttest，最后执行一个错误的命令 test。如果你用的是 Windows 操作系统，请把“r'mkdir qttest'”改成“r'md qttest'”，其他命令也是换成你对应的系统的命令行的命令。在这里其实每个命令的执行都有状态码的返回，一般 0 都是正常的，非 0 都是有问题的。所以最后一行的脚本会有一个非 0 的状态码返回，在实际项目的案例里，大家可以对状态码进行判断。至于命令的结果的返回，笔者还没尝试出 Evaluate 的相关用法，不过后面有一个 OperatingSystem 的测试库里会有关键字可以得到命令的结果返回。脚本见表 7-2-18。

表 7-2-18

${sys}	Evaluate	sys.platform	sys
Pass Execution If	'${sys}' <> 'darwin'	非Mac系统请自己修改命令	
${cmd}	Evaluate	os.system(r'mkdir qttest')	os
${cmd}	Evaluate	os.system(r'ls')	os
${cmd}	Evaluate	os.system(r'rm -rf qttest')	os
${cmd}	Evaluate	os.system(r'test')	os

7.3 String

String 这个库从名字就可以看出，它主要是用来操作字符串的。其实前面的 Evaluate 的例子里也有一部分字符串操作，这些操作同样可以在 String 库里找到，只不过要加载 String 库才能使用。比如 Replace String，用 Evaluate 写是下面这样，脚本见表 7-3-1。

表 7-3-1

${str}	Evaluate	'${str}'.replace('o','h')

如果用 Replace String 来写就是这样，脚本见表 7-3-2。

表 7-3-2

${replace}	Replace String	${str}	o	h

这两种选择哪种都可以，具体看案例的情况，如果只是偶尔使用，推荐 Evaluate。这样就不用为了使用几次而额外加载一个测试库了，如果案例里有很多字符串的处理，建议

加载 String 库，使用里面的关键字。

String 库的测试案例准备了 3 个，我们分别看一下。

7.3.1　Convert

转换这里首先是大小写的转换，这个都很好理解。后面是把字符串 Encode 成对应编码的 Bytes，然后再用 Decode 转回字符串。脚本见表 7-3-3。

表 7-3-3

${str}	Set Variable	Hello World	
${lower}	Convert To Lowercase	${str}	
${upper}	Convert To Uppercase	${str}	
${a}	Set Variable	中文	
${utf8}	Encode String To Bytes	${a}	UTF-8
${gbk}	Encode String To Bytes	${a}	GBK
${utf8cn}	Decode Bytes To String	${utf8}	UTF-8
${gbkcn}	Decode Bytes To String	${gbk}	GBK

7.3.2　Line

这个案例主要是介绍 String 库对多行文本的处理，哪里会有多行文本呢？比如文件内容、命令行返回结果等，所以这里的关键字其实要结合获取文件内容，或者获取命令行返回值使用比较合适，在后面的小节中会有这两个的例子。

在这里为了单纯介绍这个库，笔者直接造了一个多行文本，大家可以看到字符串里使用了“\n”这个换行符，所以后面才能 Get Line Count 获取字符串行数，然后再把文本内容 Split To Lines 按行拆分成单行的文本 List。Get Line 可以指定返回某一行的内容，而 Get Lines Containing String 可以指定返回包含某个字符串的行，如果是多行那就又需要 Split 了。脚本见表 7-3-4.

表 7-3-4

${str}	Set Variable	hello\nworld\nqitao\nrobot	
${linecount}	Get Line Count	${str}	
${lines}	Split To Lines	${str}	
${line}	Get Line	${str}	1

续表

${linecontain}	Get Lines Containing String	${str}	t
${lines}	Split To Lines	${linecontain}	

7.3.3 String

前面我们把多行的文本拆分成单行的文本，其实是在于单行文本能让我们更方便地去操作字符串处理。这个案例里就主要是字符串处理的例子了，像前面提到的 Replace String，Remove String 其实也可以用 Replace String 来完成，只要把要替换成的值换成${EMPTY}就行了。Split String 就是用分隔符拆分字符串，比如这里不写就是默认用空格拆分。Split String From Right 就是从右边拆分，如果不加第三个参数，其实这两个关键字没什么差异，如果加上了第三个参数 max_split，就是最大拆分次数。比如这里写的是 1，也就是从右边开始用空格拆分，最多拆分 1 次。Fetch 的 2 个关键字其实和 Split 类似，但是 Split 出来是个 List，而 Fetch 是字符串。比如 Fetch From Right，就是从右边向左边找到第一个指定的字符“o”，然后 Fetch 它的右边的字符串，Left 就是从左边向右边找。

Get Substring 这个用来截取字符串，其实还有更简单的截取字符串的方法，比如${str[6:]}就和我们使用的效果一样。最后一行的关键字 Generate Random String 就是用来生成随即字符串的，前面在讲 Evaluate 的时候用了很多行脚本实现，在这里只需要一个关键字就可以了。脚本见表 7-3-5。

表 7-3-5

${str}	Set Variable	hello world qitaos		
${replace}	Replace String	${str}	o	h
${remove}	Remove String	${str}	wo	
${split}	Split String	${str}		
${splitright}	Split String From Right	${str}		1
${splitchar}	Split String To Characters	${str}		
${fetchright}	Fetch From Right	${str}	o	
${fetchleft}	Fetch From Left	${str}	o	
${sub}	Get Substring	${str}	6	
Log	${str[6:]}			
${gen}	Generate Random String	4	[UPPER]	

7.4　Collections

Collections 库可以翻译为集合，List 和 Dictionary 都可以算是集合，在 Python 里主要有 List 列表、Dictionary 字典、Tuple 元祖和 Set 集合等。在 Collections 库里主要是 List 和 Dictionary 的关键字。

7.4.1　List

Convert To List 可以把 Tuple 或者可遍历的列表对象转换成 List；Append To List 是在 List 后面加上新的元素；Insert Into List 则是在 List 中间插入元素；Get From List 是通过 Index 获取 List 的一个元素；Get Index From List 正好相反，获取某个元素在 List 的 Index。这里只列了几个常见的关键字，还有更多的关键字可以看一下 github 上的例子。脚本见表 7-4-1。

表 7-4-1

${tuple}	Evaluate	(u'a',u'b',u'c',u'd')	
${conlist}	Convert To List	${tuple}	
Append To List	${conlist}	e	f
Insert Into List	${conlist}	1	q
${get}	Get From List	${conlist}	-1
${getindex}	Get Index From List	${conlist}	f

7.4.2　Dictionary

字典其实在前面的章节里已经介绍过了，并且也有一些例子了，比如 Create Dictionary 创建字典。Set To Dictionary 其实和创建字典是差不多的作用，只不过如果有的 key 已经存在，这个关键字就会覆盖 key 的值。Remove From Dictionary 和 Keep In Dictionary，前者是去掉字典里的 key，后者是保留字典里的 key。脚本见表 7-4-2。

表 7-4-2

${dict}	Create Dictionary	a=1	b=2
Set To Dictionary	${dict}	a=3	c=4
Log Dictionary	${dict}		
Remove From Dictionary	${dict}	a	

续表

Log	${dict}		
Keep In Dictionary	${dict}	b	
Log	${dict}		

7.5 OperatingSystem

前面的小节里其实提到过 OperatingSystem 库，基本上和操作系统有关的关键字都在这里。比如环境变量、文件、文件夹、文件路径等，后面的案例也基本上是按这几个方面做的。

原本这里还有操作进程的一些关键字，不过 RF 将 Process 独立创建了一个测试库，所以进程的操作都推荐使用 Process 库。所以 OperatingSystem 里的操作进程的关键字这里没有列出来。

7.5.1 Env

首先介绍了一下 3 个 Run 的关键字，执行的命令都是“df –h”，也就是查看磁盘空间使用情况，在 Mac 操作系统或 Linux 操作系统上都可以，Windows 操作系统请自己更换成 Windows 的命令。这 3 个关键字，第一个是只返回结果；第二个是只返回状态码；第三个则是状态码和结果都返回。在后面的就是对环境变量进行的一些操作了，比如获取环境变量、设置环境变量等，脚本见表 7-5-1。

表 7-5-1

${output}	Run	df -h
${rc}	Run and Return RC	df -h
${rcandoutput}	Run and Return RC and Output	df -h
${path}	Get Environment Variable	PATH
Set Environment Variable	TEST	ride
${test}	Get Environment Variable	TEST
Append To Environment Variable	TEST	robot
${envs}	Get Environment Variables	
Remove Environment Variable	TEST	
Log Environment Variables		

7.5.2　File

这个部分是介绍对文件的操作方法，首先是 Create File，指定文件内容为“中文内容”，然后通过 Get File 可以获取到文件的内容；Get File Size 获取文件大小；Grep File 是在文件内容中搜索特定关键字的行；Touch 是类似于 Linux 的 Touch，作用就是创建一个空文件或者修改文件的时间戳；最后的 Remove Files 就是删掉这些文件。脚本见表 7-5-2。

表 7-5-2

Create File	${CURDIR}${/}file.txt	中文内容	
${file}	Get File	${CURDIR}${/}file.txt	
${file-size}	Get File Size	${CURDIR}${/}file.txt	
Append To File	${CURDIR}${/}file.txt	\n自动化测试\n测试指南	
${grep}	Grep File	${CURDIR}${/}file.txt	测试
Touch	${CURDIR}${/}touch.txt		
Remove Files	${CURDIR}${/}touch.txt	${CURDIR}${/}file.txt	

7.5.3　Directory

文件操作介绍完之后就是目录操作了，这里 Create Directory 就是在指定路径创建目录，Copy/Move/Remove/Empty Directory 分别是复制/移动/删除/清空目录的作用。List Directory 等同于 ls 或者 Windows 的 dir 命令。Count Directories In Directory 就是计算目录里有多少个子目录。脚本见表 7-5-3。

表 7-5-3

Create Directory	${CURDIR}${/}dir1	
Create Directory	${CURDIR}${/}dir1${/}subdir	
Copy Directory	${CURDIR}${/}dir1	${CURDIR}${/}dir2
Move Directory	${CURDIR}${/}dir2	${CURDIR}${/}dir3
${list}	List Directory	${CURDIR}
${countdir}	Count Directories In Directory	${CURDIR}
Remove Directory	${CURDIR}${/}dir3	
Empty Directory	${CURDIR}${/}dir1	
Remove Directory	${CURDIR}${/}dir1	

7.5.4 Path

Join Path 比较简单，就是把后面的几个参数都拼起来。而 Join Paths 是在 basepath 的基础上，和后面的每一个分别拼装，如果碰到以“/根目录”开头的，则拼装结果还是原先的“/根目录”开头的内容。Normalize Path 是将路径正常化，也就是真实路径，像 ${CURDIR}..${/}7.3-String 这个的真实路径，就是当前目录的上一级目录下的 7.3-String 目录这一个路径。Split Path 是将路径拆分开，从右起向左找到第一个路径分隔符，然后拆分开；Split Extension 是以扩展名来进行拆分；最后的 Get/Set Modified Time 是用来获取或设置文件最新修改时间的；Get 可以获取文件或目录的最新修改时间；Set 只能设置文件的最新修改时间。脚本见表 7-5-4。

表 7-5-4

${path}	Join Path	${CURDIR}	test	
${paths}	Join Paths	${CURDIR}	/usr	test
${normalpath}	Normalize Path	${CURDIR}..${/}7.3-String		
${split}	Split Path	${CURDIR}		
${split}	Split Extension	${CURDIR}${/}sample.txt		
${time}	Get Modified Time	${CURDIR}		
Set Modified Time	${CURDIR}${/}sample.txt	NOW - 1day		
${time}	Get Modified Time	${CURDIR}${/}sample.txt		

7.6 Process

这个库主要就是和进程有关的操作了。

首先这里用了 Run Process 的两种不同用法。一个是直接获得返回结果对象；另一个是通过 Get Process Result 获得返回结果对象，但是返回的对象用法都是一样的。

后面就是用 Start Process 来启动不同的进程，并加上 alias 别名，通过 Switch Process 可以切换进程，最后 Terminate All Processes 用来终结所有通过前面脚本打开的，还活跃着的进程。脚本见表 7-6-1。

表 7-6-1

${result}	Run Process	python	-c	print "robot"
Log	${result.stdout}			

续表

Run Process	python	-c	print "rf"	alias=rf
${presult}	Get Process Result	rf		
Start Process	python	alias=py2		
Start Process	ping	alias=ping		
Switch Process	py2			
Terminate All Processes				

7.7　XML

XML 这个库从名字就可以看出来，这是用来处理 XML 的。前面做接口测试的例子里，其实也有一部分 XML 的例子了。

这里首先用 Parse Xml 来将一段文本信息处理成 XML 对象。用 Get Element 可以获取 XML 的节点元素，Add Element 则可以添加一个节点。

以这个节点为例（<auto id='4'>qitao</auto>），Element 的 Tag 是 auto，Attribute 是 id='4'，Text 是 qitao。明白了这几个值的含义，接下来的关键字里对 Element 的 Tag、Attribute、Text 等处理，你就能明白它的作用是什么了。最后的 Clear Element 就是清理掉所有的节点，保留了根节点<xml />。脚本见表 7-7-1。

表 7-7-1

${XML}	Parse Xml	<xml><robot>test</robot></xml>	
${element}	Get Element	${XML}	robot
Add Element	${XML}	<auto id='4'>qitao</auto>	
${text}	Get Element Text	${XML}	auto
Set Element Tag	${XML}	rf	robot
Set Element Text	${XML}	settext	xpath=rf
Remove Element Attribute	${XML}	id	auto
Clear Element	${XML}		

7.8 其他测试库

- DateTime：处理日期和时间的一些关键字；
- Dialogs：在案例运行的过程中弹出一些对话框，需要人工来操作的，不过这个和自动化测试的思路不符；
- Screenshot：用来对整个桌面截图的一些关键字；
- Telnet：在终端上通过 telnet 命令，进行远程连接操作的一些关键字；
- Remote：这个略微有点绕，它就是在某个 Server 上将一个 Library 通过 remoteserver 共享出来。然后这个库加载的时候，使用这个 Server 对应的 IP 和端口，就可以使用里面的方法了。换个思路，比如说 Selenium2Library 需要安装才能使用，但是如果有个 Server 将 Selenium2Library 通过 remoteserver 共享出来，那么就可以使用它里面的方法，而不需要安装了。

7.9 小结

这一章介绍了常见的几个测试库，并提供了一些例子以便大家更好地理解这些测试库如何使用。还有一部分测试库并未一一列出，如果需要了解它们更全面的关键字说明，首推官方网站的文档，地址如下：

http://robotframework.org/robotframework/

8

第 8 章
持续集成自动化测试

现在越来越多的项目组开始引入敏捷，其中对于我们的自动化测试也有了一些新的要求。至少要把我们做的自动化案例能够持续地跑起来，能够在敏捷模式下良好地运转起来。

8.1 Jenkins 安装与配置

8.1.1 Jenkins 简介

Jenkins 目前应该是大家使用的比较多的一个持续集成的工具了，不管是开发测试部署，还是代码扫描等。很多工作都可以放到 Jenkins 来进行，最终实现一个完整的流程。开发人员提交代码后，Jenkins 上的 job 就会开始“自动编译打包→自动部署→单元测试/代码扫描→自动测试”。单元测试这边一般是 Junit 之类，代码扫描算是笼统的称呼，实际上可以有很多，比如 checkstyle、findBugs、sonar 等。在 Jenkins 上最基本的就是 job 了，通过 job 之间建立关联就可以形成前面的这个流程，并且基本上很少需要人工干预。此外就是

各种各样的插件，使得 Jenkins 这个平台能够整合很多强大的功能进来，满足用户的需要。

8.1.2 安装 Jenkins

Jenkins 下载地址：

http://mirrors.jenkins-ci.org/

进入上面的地址，页面里有 Jenkins 的下载目录，各种平台的各种 release 的版本，大家根据自己需要来下载，笔者个人比较推荐下载 war 包，如果对版本没特别的要求，就进入 war 的第一行 release 里面找最新版本用吧。

为什么推荐 war 包呢？因为它比较简单方便。如果你有 Tomcat，那么把 war 包放到它的 webapps 目录里就可以了，直接启动你的 Tomcat 即可。

通常访问路径是：http://localhost:8080/jenkins。

如果你没有 Tomcat，那就更简单了，这里不会教你怎么去安装 Tomcat，因为 Jenkins 的 war 包里内置了 jetty，直接用命令行 java -jar jenkins.war 就可以启动 Jenkins 了。如图 8-1-1 所示。

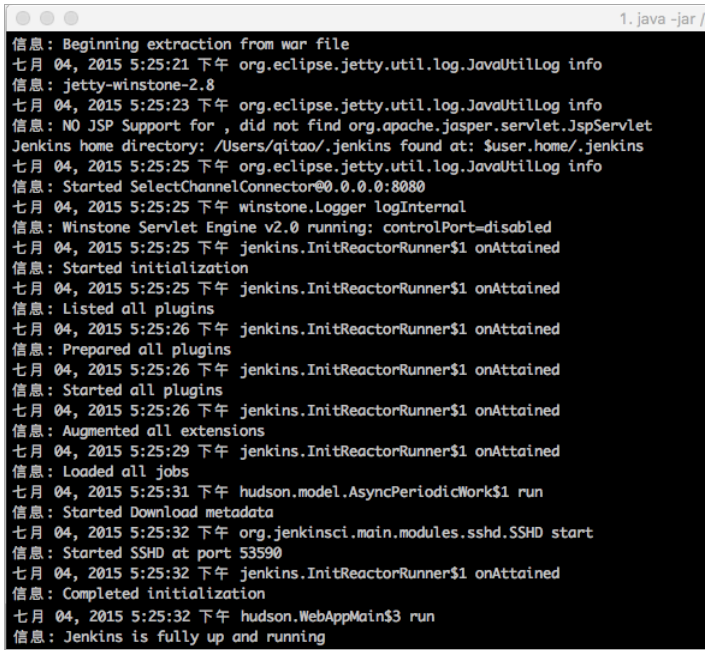

图 8-1-1

推荐这样的方式，是因为这样直接就可以让其他人访问了，可以看到图中的第 8 行：Started SelectChannelConnector@0.0.0.0:8080。

这里的 Jenkins 地址“0.0.0.0:8080”，这就是开启服务的 IP 和端口了。而如果是在 Tomcat 里还要修改配置 IP 为“0.0.0.0”才行。所以这样直接用 war 包的话方便了不少。

当你看到这行信息：“Jenkins is fully up and running”，就代表 Jenkins 的服务已经启动了。

在 Chrome 里访问 Jenkins 首页，速度会快些，在 Chrome 中打开 Jenkins 地址：http://192.168.1.2:8080/。前面日志里的 IP 地址“0.0.0.0”表示是以本机真实 IP 地址“192.168.1.2”作为服务器 IP 地址，你也可以用 http://localhost:8080/ 或者 http://127.0.0.1:8080/，都可以打开 Jenkins 首页，如果是给其他人访问，就是用真实 IP 地址访问了。打开 Jenkins 的首页界面，如图 8-1-2 所示。

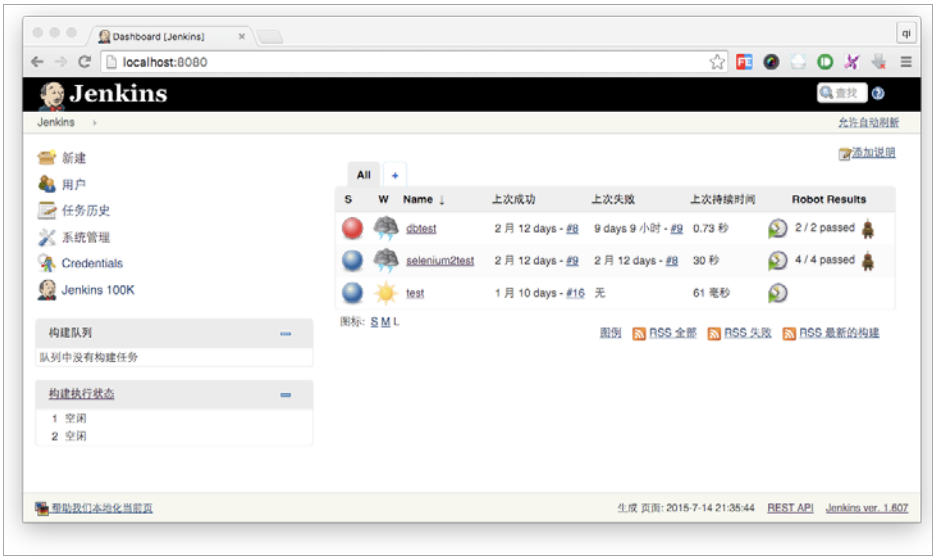

图 8-1-2

8.1.3　配置节点

启动 Jenkins 服务的这台机器在 Jenkins 里叫做 master，而其他的连到 master 上的机器（物理机、虚拟机都可以）都是 slave，也叫节点。如果在 master 上运行案例，通常是隐藏进程运行的。所以，如果你的案例是有 UI 自动化测试的，最好添加一个 slave 来执行，

否则很可能会执行失败。如果有多台机器需要接入 Jenkins 来执行案例，那更要加一下 slave。

单击“系统管理”选项，然后单击“管理节点”选项，看到的界面，如图 8-1-3 所示。

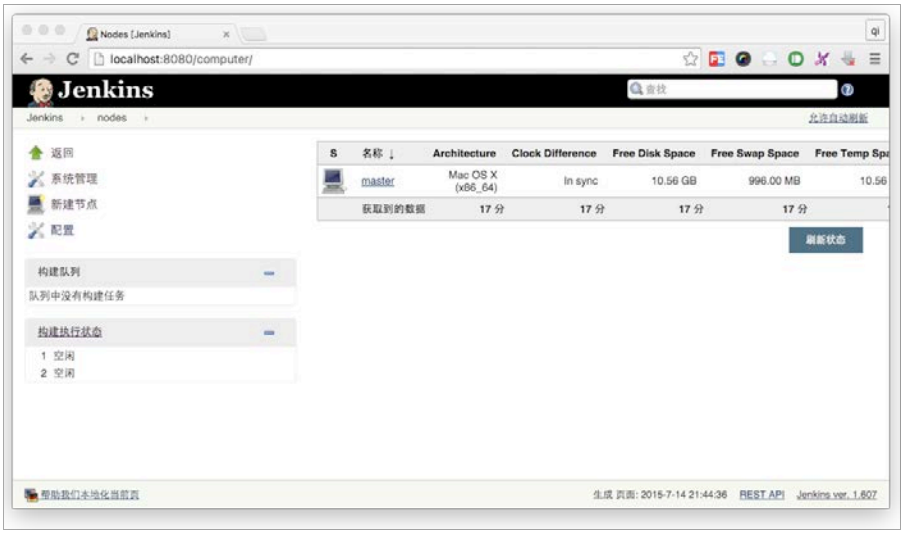

图 8-1-3

选择左边的“新建节点”选项，输入节点名称 test，选择“Dumb Slave”单选按钮，如图 8-1-4 所示。

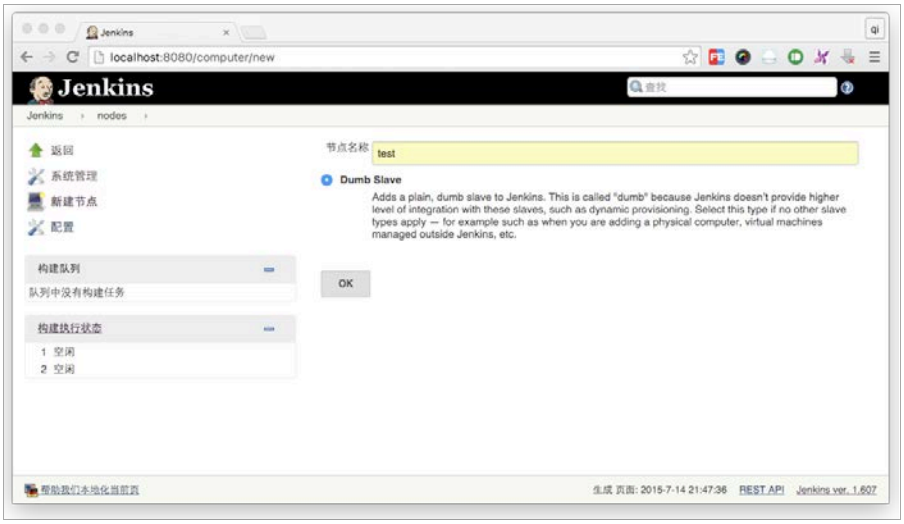

图 8-1-4

如果已经添加过节点了，可以选择“复制现有节点”单选按钮，如图 8-1-5 所示。

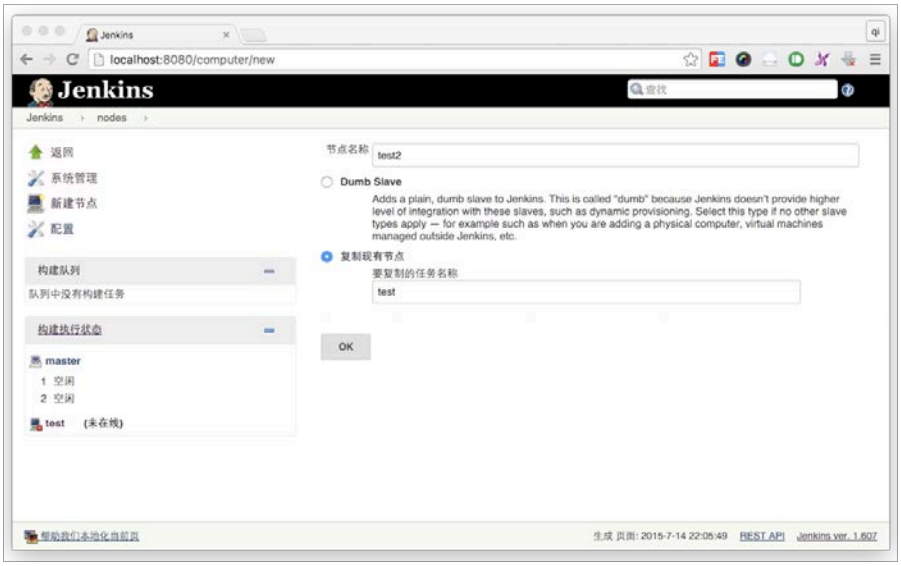

图 8-1-5

单击“OK”按钮之后，进入节点设置界面，如图 8-1-6 所示。

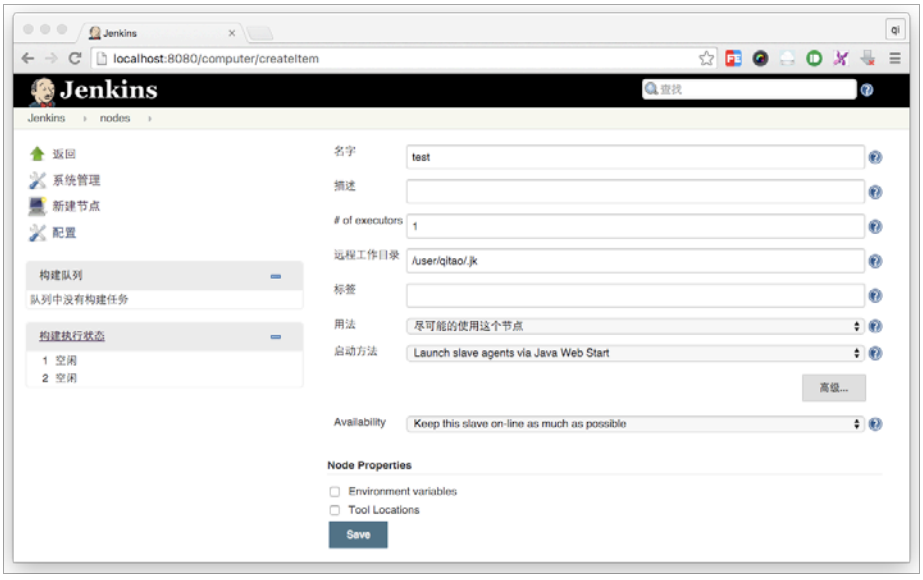

图 8-1-6

这个页面上的一些配置项，具体说明如下。

- # of executors：这个节点上最多可以并行执行几个任务，一般如果是 UI 测试的 Job，最好只允许 1 个，其他的可以设置为多个；
- 远程工作目录：要设置这个节点的工作目录的路径，在这个节点上运行的 Job 都会保存到工作目录里，这个必须要设置；
- 标签：等同于测试案例的 Tag，可以设置 Job 在某些 Tag 的执行机上运行，如果指定固定的某个 slave，可能上面有 Job 在运行了，就要等待了。但是如果有一批 slave，使用同一个 Tag，只要配置 Job 要运行在这个 Tag 上，那么哪台 slave 空闲了就会到哪台上执行。推荐设置，这里先设置为 robot；
- 用法：是指这个节点如何被使用，有两个选项，尽可能的使用这个节点和只允许运行绑定到这台机器的 Job；
- 启动方法：Launch slave agents via Java Web Start，一般选择这个就可以了，其他几项都不太适合我们；
- Availability：设置节点在线的时间。一般默认第一个就行了。

再往下面还可以设置这个节点的环境变量或者工具位置，这里不做重点介绍。

都设置好之后，单击“Save”按钮，这个节点才算生成了。此时可以看到所有的 slave 节点列表，如图 8-1-7 所示。

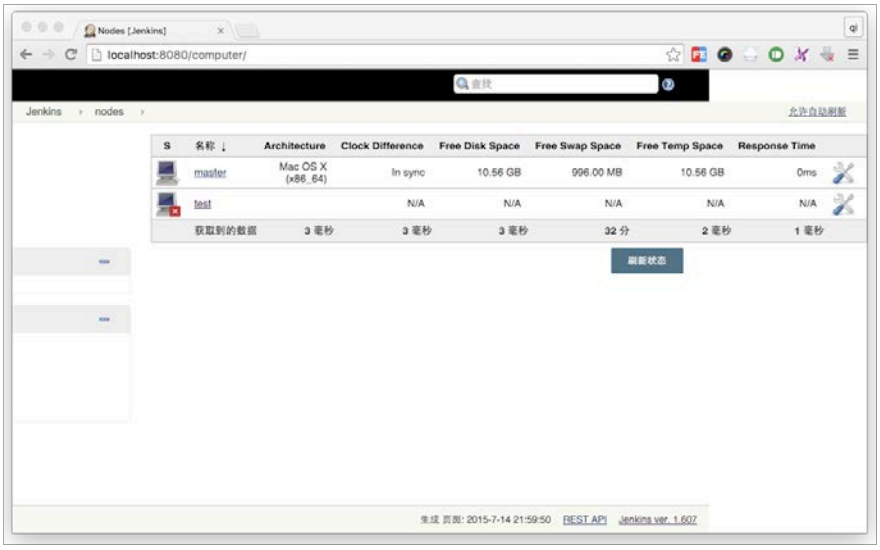

图 8-1-7

单击刚刚新增节点的名称链接，可以看到节点详情页面，如图 8-1-8 所示。

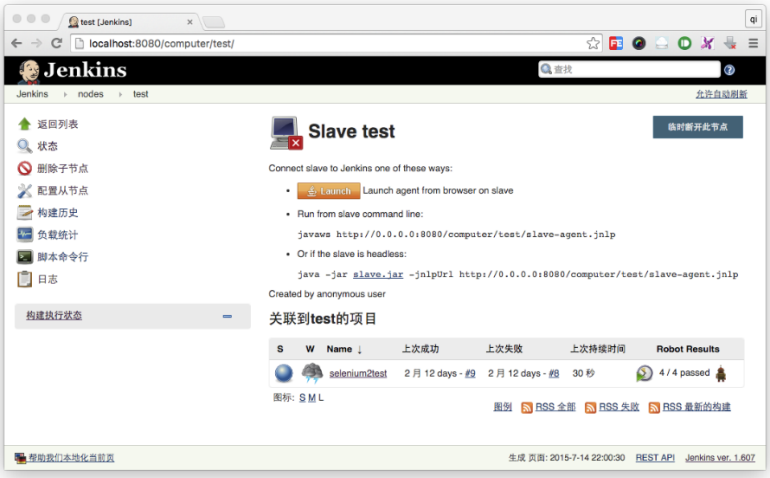

图 8-1-8

这时 slave 图标上的“叉”表示它还是 off line 的。这里提供了几种启动 slave 的方法，笔者个人觉得第二种命令行执行 javaws 的方式比较简单。在你需要启动 slave 的机器上执行这个命令就可以了，不过要把“0.0.0.0”换成真实 IP 地址。

或者选择“系统管理→系统设置”选项，配置一下 Jenkins URL 为 http://192.168.1.2:8080/，然后再到 slave 的页面，看到的就是下面的界面，如图 8-1-9 所示。

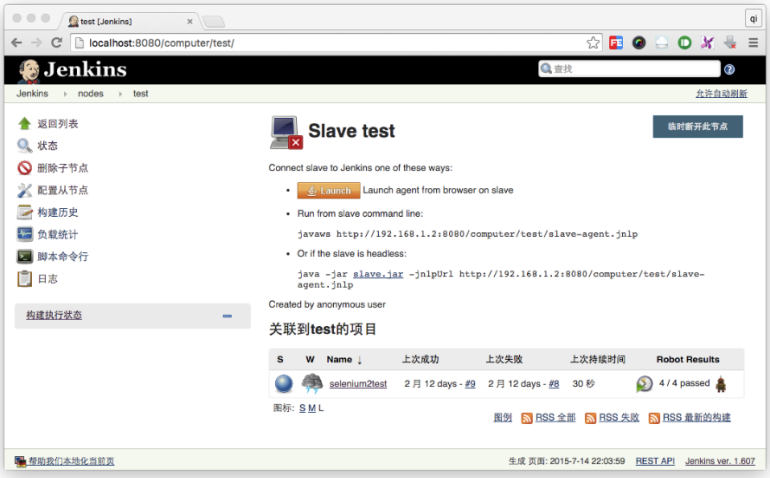

图 8-1-9

8.1.4 安装插件

接下来要安装插件了，基本上插件的安装就两种方式，这里分别介绍一下。

回到 Jenkins 首页，进入系统管理，单击“管理插件”选项，进入到“插件管理”界面。这里有“可更新”、“可选插件”、“已安装”和“高级”4 个选项卡。

1. 可更新

这里会列出已经安装的插件中，哪些有了新版本。可以勾选要更新的插件，单击“下载待重启后安装”按钮，这时就会进入下载更新界面，界面下方的选项可以勾选。安装完成后重启 Jenkins（空闲时），如图 8-1-10 所示。

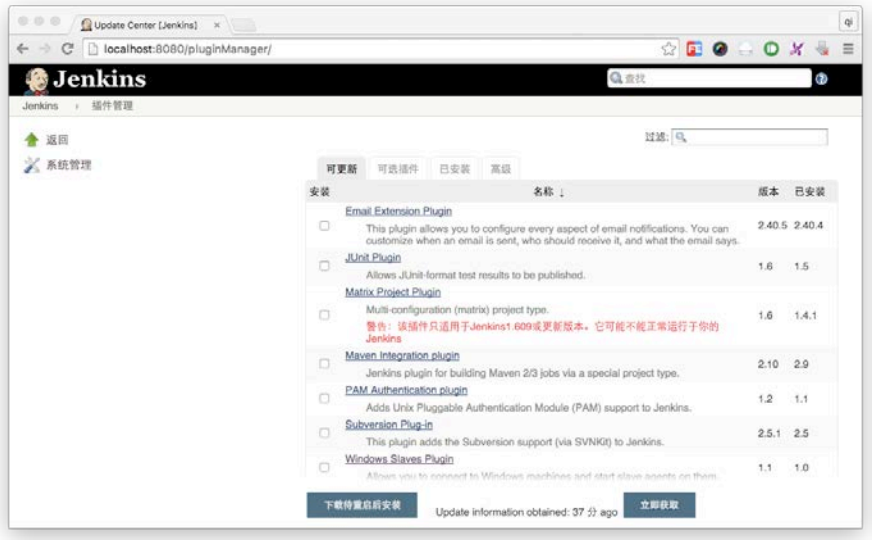

图 8-1-10

一般 Jenkins 都是把插件下载好，然后等下一次重启后安装生效。如果是公司使用的 Jenkins，一般重启的时候都会比较谨慎，避免影响到正在执行的 Job。如果是自己小团队的 Jenkins 那就影响不大了，只要不影响使用，选择空闲时重启就可以了。

2. 可选插件

可选插件里会列出所有可以选择安装的插件，所以最方便的安装插件就是在可选插件这里了。可以同时勾选多个需要安装的插件，单击“直接安装”或者“下载待重启后安装”按钮即可，如图 8-1-11 所示。

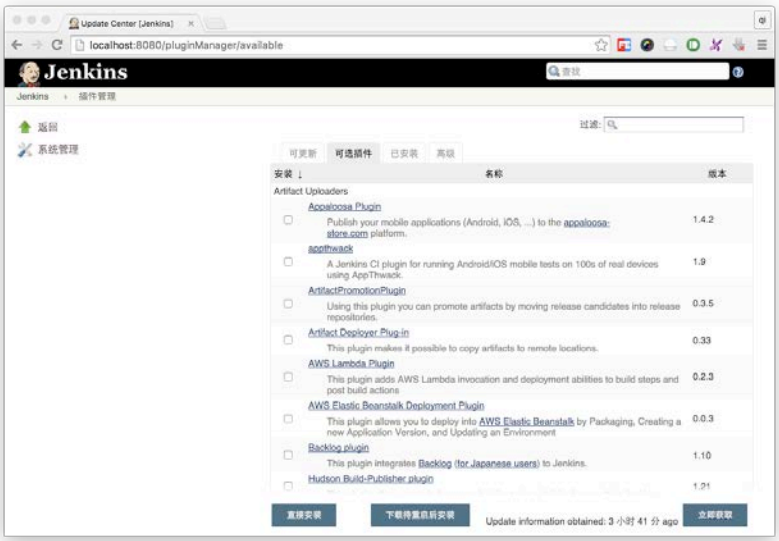

图 8-1-11

3. 已安装

这里会列出已经安装的插件列表，可以单击“降低版本（降到）”或者“卸载”按钮，如图 8-1-12 所示。

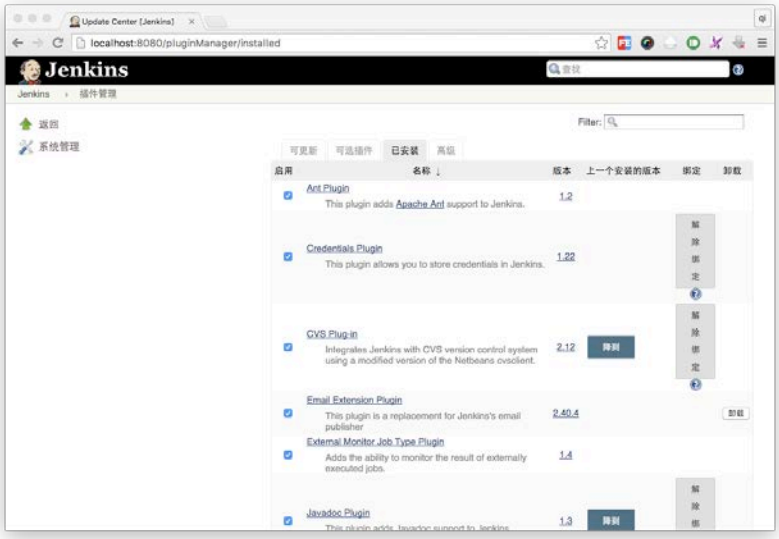

图 8-1-12

4. 高级

如果 Jenkins 在防火墙里，不方便直接访问互联网，这里可以设置代理，通过代理让 Jenkins 来下载和安装插件。

当然，如果实在没有代理，那就只能在其他地方下载好插件文件，再上传插件这里，将插件文件上传到 Jenkins 里并进行安装。同样是重启 Jenkins 就可以完成安装了，如图 8-1-13 所示。

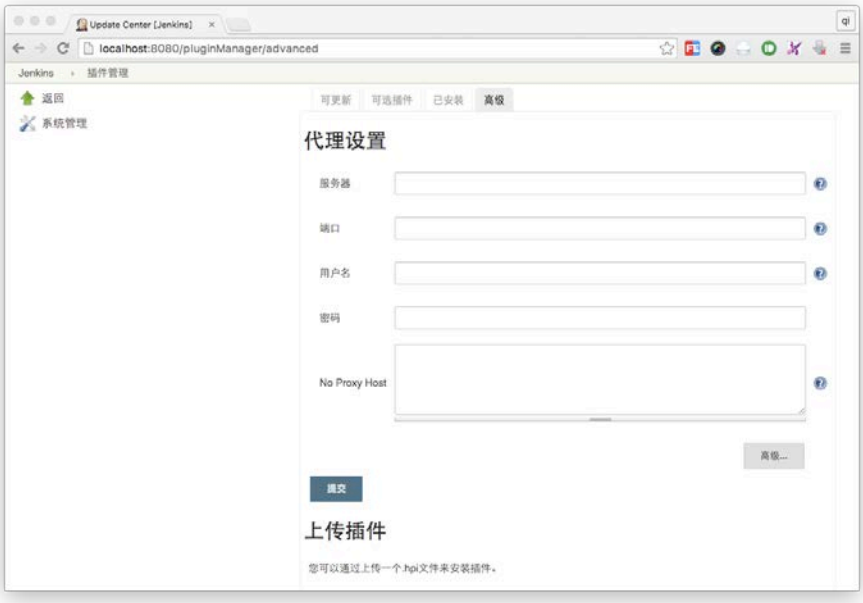

图 8-1-13

这里主要用到 Robot Framework 的 Jenkins 插件，它的下载路径：

http://mirrors.jenkins- ci.org/plugins/robot/

RF 的插件版本选最新的 1.6.1 就行了（如果有更高的版本也可以选择）。可以选择到这里找下载文件，或者直接在可选插件里搜索 Robot Framework 就可以找到了。

8.2 Jenkins 上执行 RF 自动化测试

前面这些基础工作都做好之后，就可以开始准备 RF 自动化测试了。当然，前提是自

动化测试案例已经准备好了，Jenkins 这里只是配置执行自动化测试的。

8.2.1　创建 Job

在 Jenkins 首页上的左上角有个“新建”选项，单击“新建”选项，进入页面，在“Item 名称”文本框中输入 free-config（不能和已有 Job 重名），如图 8-2-1 所示。

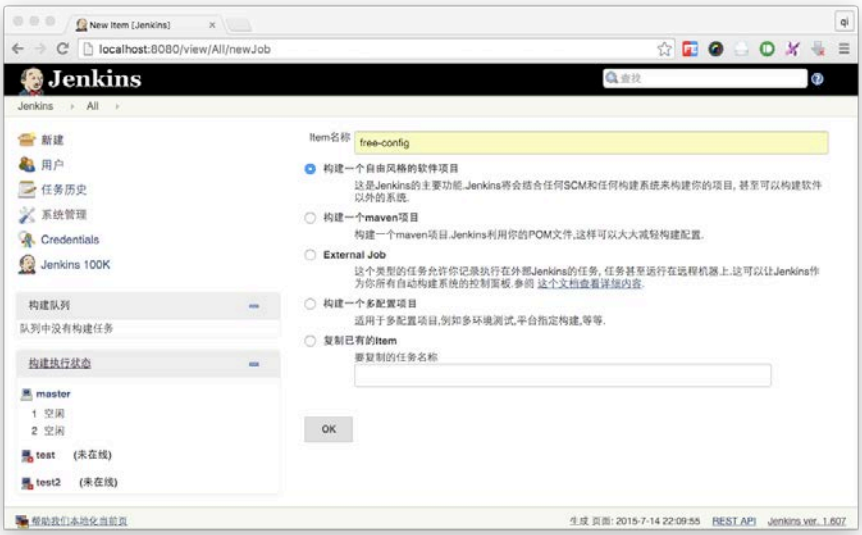

图 8-2-1

Job 类型选项有 4 个，这里简单介绍一下：

- 构建一个自由风格的软件项目：自由风格，即可以随意搭配任何插件，这种 Job 用的最多；
- 构建一个 maven 项目：基于 maven 的 Job；
- External Job：这个在说明上介绍，可以记录外部的 Jenkins 上的任务，任务可以运行在远程机器上；
- 构建一个多配置项目：这个不推荐新手使用，光是多配置就会把你搞晕。简单地说，就是它可以把一个 Job 在多个 slave 上同时执行，每个 slave 上可以有一套配置，这就是多配置。笔者能想到的就是在不同的 slave 上使用不同的浏览器来执行测试；

- 复制已有的 Item：同一个系统的案例，建新版本的 Job 时，可以考虑直接复制老版本的 Job，或者是配置类似的，复制一份再修改。

一般情况下，推荐使用第一个自由风格和最后一个复制已有的 Item。其他几个大家根据自己的需要选择。

8.2.2 配置 Job

Job 的配置先介绍自由风格的，多配置的后面会提一下。可以配置的内容很多，先介绍一下重要的配置，其他的会最后一起介绍。

1. 丢弃旧的构建

丢弃旧的构建界面，如图 8-2-2 所示。

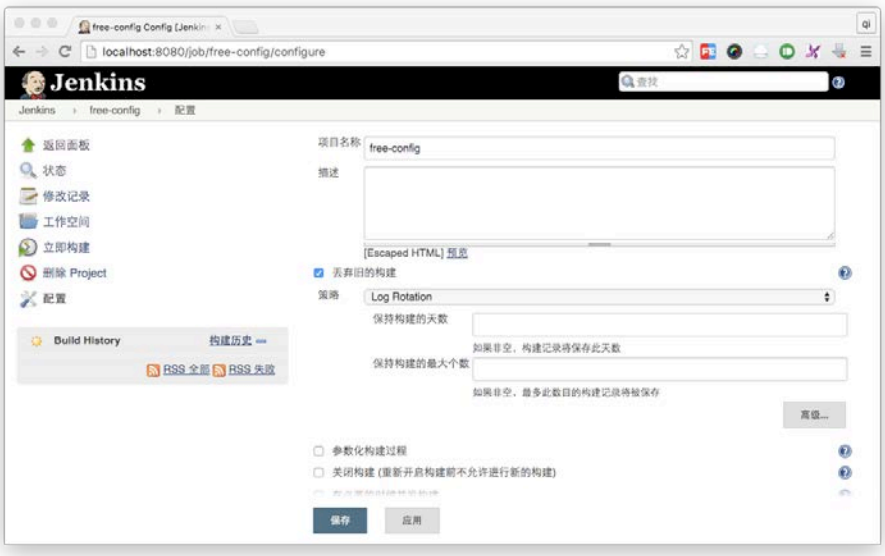

图 8-2-2

因为每构建一次 Job 就会有一次记录，如果构建过多，可能也会占用很多硬盘空间。因此，这个设置就是可以让我们控制如何丢弃旧的构建。勾选“丢弃旧的构建”复选框后，有两个选项：

- 保持构建的天数：保留多少天内的构建，超过这个天数的就会丢弃；
- 保持构建的最大个数：最多保留多少个构建。

这样可以降低一些 master 的存储和 Job 的构建历史记录，根据自己需要进行设定吧。

2. 参数化构建过程

参数化构建过程界面，如图 8-2-3 所示。

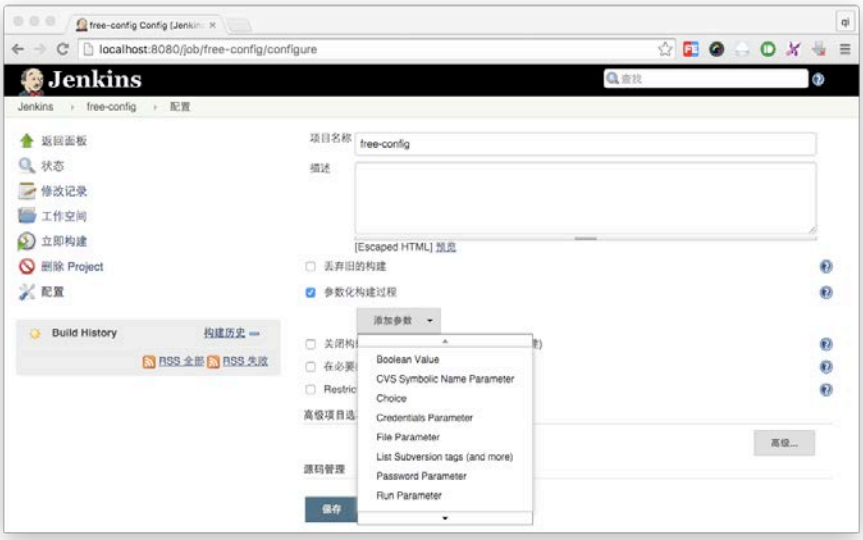

图 8-2-3

可以增加一些不同类型的参数，在构建时通过修改这些参数的值，来方便地改动一些配置。比如说目前在用的 App 的打包 Job，就会加上版本号、环境、日志开关等参数，可以通过运行时简单地修改参数值，就可以打出指定版本号、测试或生产环境、日志开关打开或关闭等满足条件的安装包。

3. Restrict where this project can be run

Restrict where this project can be run 界面，如图 8-2-4 所示。

勾选这一项的复选框，就是指定这个 Job 只能在某些特定的 slave 上执行。可以用 slave 的名称或者标签。前面介绍 slave 的时候提到了，推荐使用标签。因为如果直接使用 slave 的名称的话，可能会在 Job 比较多的时候造成拥堵，特别是在 slave 设置了最大并行数量比较少的情况下。而使用标签，那么只要是带有这个标签的 slave，哪个空闲哪个就会安排 Job 执行，这样可以使资源充分利用，效率也会比较高。

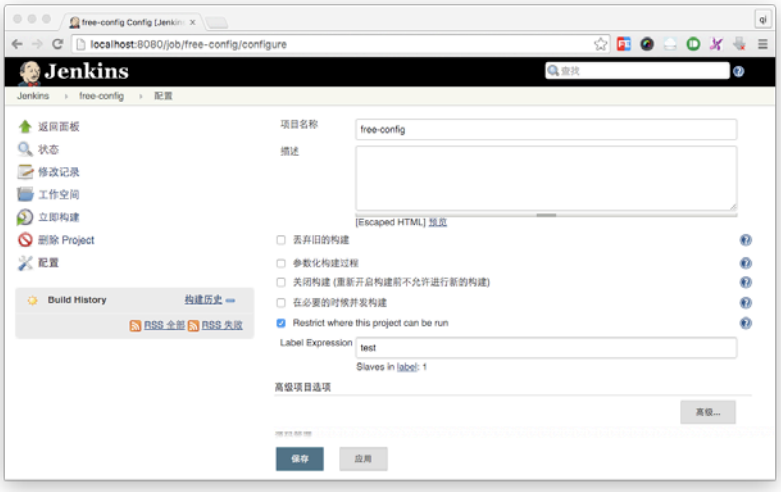

图 8-2-4

4. 源码管理

源码管理界面，如图 8-2-5 所示。

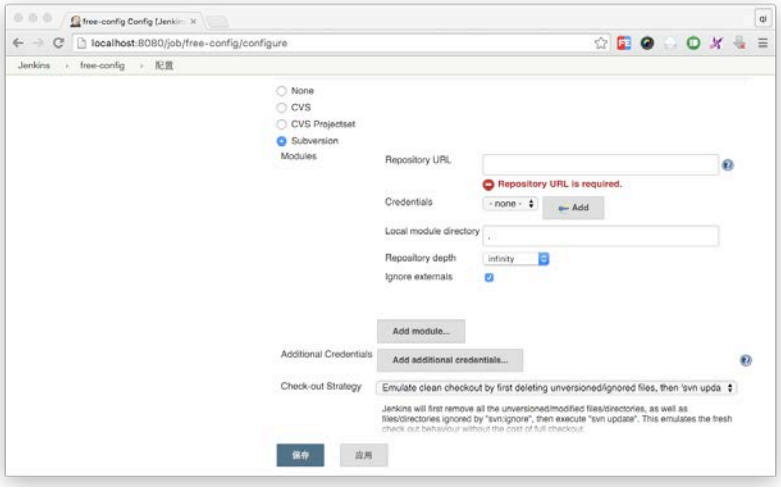

图 8-2-5

有很多种源码管理工具可以选择，如果没有的话可以下载相应的插件。默认已有的几种，cvs、cvs Projectset 和 svn。如果使用的是 git，则需要安装 git 插件。

为什么要用源码管理？推荐大家的案例使用版本管理，这样在案例更新后，Job 就会

更新最新的案例来运行。如果没有版本管理也不是不行，就是要人工去做更新的动作，相对比较麻烦。

这里选择 subversion，也就是 svn，可以看到几个配置项，这里介绍一下必须配置的几个配置项。

- Repository URL：svn 库的地址；
- Local module directory：下载到本地的路径，相对路径，根目录是工作区路径；
- Credentials：新版的有这个，就是 svn 的认证用户密码设置，以前的版本隐藏在帮助链接里；
- Check-out Strategy：签出策略，里面有几个选项，推荐使用第三项“Emulate clean checkout by first deleting unversioned/ignored files, then 'svn update'”，这个选项会把一些不在版本控制的文件先删除掉，然后再进行 svn 更新，避免这些文件影响了签出。因为如果使用了构建参数修改环境什么的，肯定会修改到一些代码文件，如果不删除这些改动，签出时有可能会有冲突出现。第 4 项也有类似的作用，其他几个大家可以查一下相关作用。

这几个配置好了，基本上就可以了。如果有多个不同 svn 库要签出的话，可以单击“Add module”按钮添加新的 svn 库配置。

5. 构建触发器

构建触发器界面，如图 8-2-6 所示。

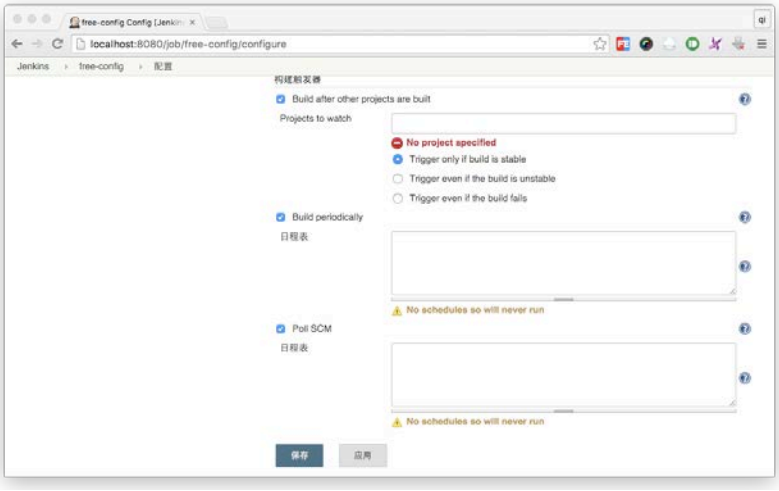

图 8-2-6

可以手动来触发一个构建，也可以通过触发器来触发其自动构建，这里的设置就要根据项目的情况自行考虑了。

（1）Build after other projects are built

这个选项是在其他 Job 构建之后，触发该 Job 构建。在构建后，操作里有一个 Build other projects，作用和这个是一样的。区别就是在这里配置了当前 Job 就是被触发的，在构建后操作配置了，当前 Job 就是触发者。

在第一个文本框里就是输入其他 Job 的名字，可以有多个 Job，用英文逗号分开，下面的 3 个选项分别是：只在构建稳定时触发、构建不稳定依然触发、构建失败依然触发。这 3 种场景也是 Job 的 3 种状态，通常默认选第一种就好了。对于自动化测试的 Job 来说，如果前面的自动部署 Job 没有成功，这里构建也没太多必要。前面提到的“自动编译打包→自动部署→单元测试/代码扫描→自动测试”这一套流程也是通过这个地方进行配置的。

（2）Build periodically

这个选项是配置一个时间表达式，让 Job 在满足时间表达式的情况下运行，比如说你可以设定每天的晚上 9 点启动 Job，或者每个小时都启动一次 Job。具体可以单击文本框右边的“问号”，就能看到时间表达式的帮助说明，虽然是英文的，但是说明的也很清晰了，这里就不详细介绍了。

（3）Poll SCM

这里配置的时间表达式和前一个是一样的，只不过这个选项是在检测到源码管理配置的版本有更新，才会触发执行。如果没有更新，即使满足了时间表达式，也不会触发 Job 的构建。这个配置通常用于开发 Job 上。比如开发人员提交了代码，这个 Job 在满足时间表达式时发现代码库上有更新，那么就会触发 Job 的构建了。

6. 构建

构建界面，如图 8-2-7 所示。

这里有很多的构建步骤，如果执行机是 Windows 操作系统，那么一般建议选择“Execute Windows batch command”来执行 cmd 命令进行构建。其他平台需要执行命令，可以选择“Execute shell”来执行命令。如果做好了 ant 或者 maven 的构建，也可以选择相应的选项。

这里增加构建步骤 Execute shell，使用 Windows 操作系统的读者要增加 Execute Windows batch command，我们构建 Robot Framework 的测试案例用到的命令是 pybot，在 Windows 操作系统下是 pybot.bat。

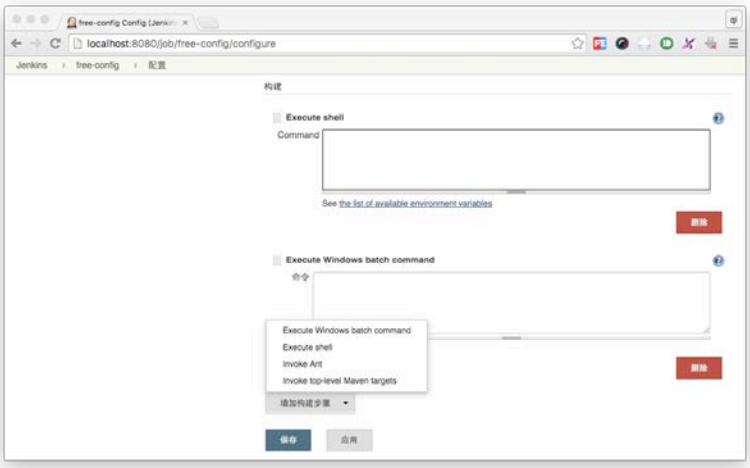

图 8-2-7

这里先写个简单的命令，完整命令行参数，大家自己在命令行输入 pybot--help 进行查看。命令如下：

```
pybot -s '*' -d ./output  ./rf-demos/fencengDemo/suite.txt
```

参数“-s”是 suite 的名称，这里选择所有的 suite 的案例，“-d”是输出结果的目录，设定在“./output”当前目录的 output 目录下，最后一个则是测试案例的 suite 文件的路径了。

7. 构建后操作

构建后操作界面，如图 8-2-8 所示。

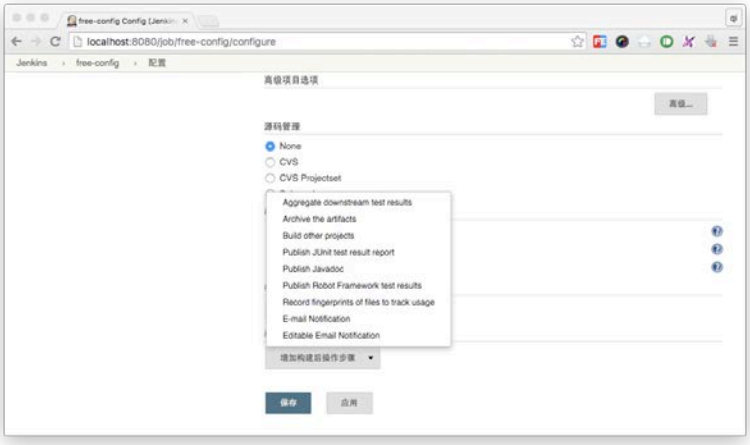

图 8-2-8

构建后的操作同样有很多，根据你添加的插件不同，可以帮你做不同的事情。比如分析测试结果，发送邮件通知等。

要增加构建后步骤是 Publishes Robot Framework test results，这个是 RF 插件的功能，界面如图 8-2-9 所示。

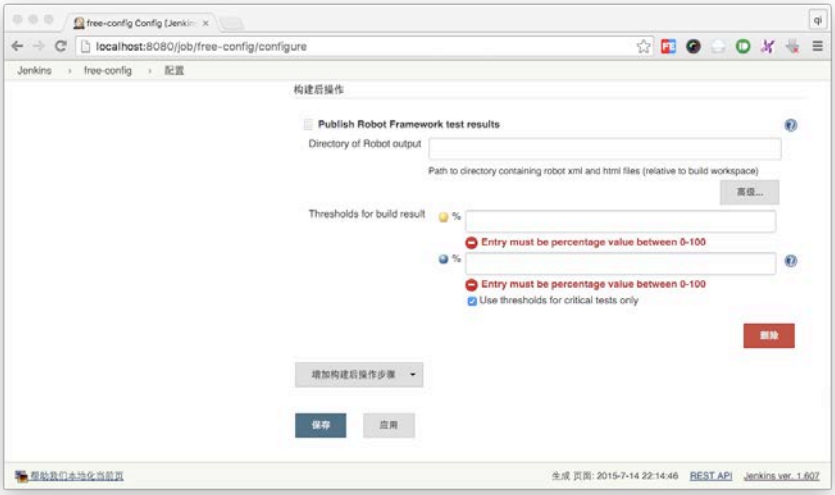

图 8-2-9

第一个 Robot output 目录默认可以为空，如果前面的 pybot 命令指定了输出目录，那么这里也要对应进行设置，否则它找不到 output.xml，就没法解析结果了。所以这个目录设置的是 RF 运行结果的 xml 和 html 文件生成的目录。比如前面我们用“-d ./output”设定了结果目录，那么在这里要设置为一样的路径。

下面的一个黄色百分比和蓝色百分比是用来设置临界值的，黄色代表 unstable 不稳定的，蓝色代表 stable 稳定的，这个就是表示 Job 的状态的。这个插件原本是期望通过这个来设定 Job 状态，不过在笔者看来，如果有案例失败了，那就应该是有问题了，而不是不稳定，所以要么全都 Pass，要么就是有失败案例。为什么说是原本期望呢？因为执行案例使用的命令行脚本，在有部分案例 Fail 后，它的状态就直接是 Fail 了，进而整个 Job 的状态就是 Fail。如果没有 Fail 那就是 100%都 Pass 了，所以这里的设置根本没什么用了，想设置的都设置成 100 就行了。

8. 其他配置

除了前面这些配置笔者认为比较重要，其他还有一些配置没有具体介绍，这里简单

列一下。

- 关闭构建：把 Job 关闭，不开启就不能构建；
- 在必要的时候并发构建：并发构建就是在当前 Job 已经在构建过程中又触发一次构建，通常情况下是串行的。在前面的构建没有结束时，不会进行下一次构建，这个选项就是允许并发构建。
- 高级项目选项：这里选项很多，因为不配置这里也没有什么影响，属于“锦上添花”的东西，这里就不细说了。

最后，如果 Jenkins 的某项功能你不会用，通常在它右边都能看到一个“问号”的图标，单击这个图标就能看到一些帮助信息，所以碰到不会用的就先看一下帮助吧。

8.2.3　控制 Job

在 Job 的首页左边，有一些操作，比较重要的是工作空间、立即构建、删除 Project、配置和构建历史，如图 8-2-10 所示。

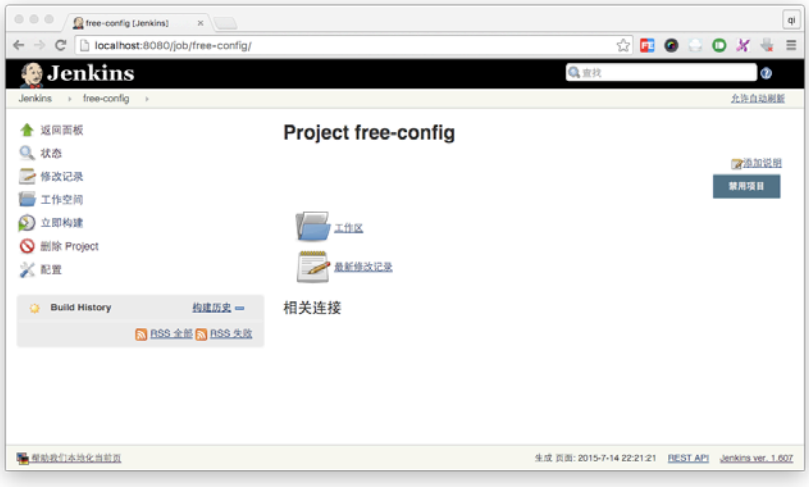

图 8-2-10

1. 工作空间

工作空间 workspace，构建时产生的文件都在这里，包括 svn 拉下来的代码库、打包好的文件、RF 的 output 目录（如果没有指定到其他地方）。所以这里相当于一个文件浏览器，你可以直接在这里查看这些文件，有必要的可以下载下来。

2. 立即构建

手动触发 Job 构建，如果配置了构建参数，则会先进入参数设置页面，确认后才会继续触发构建。对于自动触发构建的情况下，构建参数都会使用默认值进行构建。

3. 删除 Project

当你不需要这个 Job 的时候，可以通过这个命令进行删除操作。

4. 配置

这里就是进入前面 Job 配置的页面，需要修改配置就从这里进去。

5. 构建历史

每一次构建都会在构建历史里出现一条记录，单击构建编号可以看到变更集等一些信息，其中 Console Output 是控制台输出，也就是构建日志。

如果是正在构建中的 Job 会有一个进度条，可以直接单击进度条直接看到构建日志。

8.2.4 RF 插件作用

RF 插件主要用来解析 RF 运行后的 output.xml 文件，将解析的结果展示出来。html 的测试报告也会放到 Robot Results 里面。

同时装了 RF 插件后，在 Jenkins 的 Job 列表里会自动新增一列 Robot Results，如果配置了 RF 插件的，并且有运行结果的，Job 会在这里显示最新的构建结果，如图 8-2-11 所示。

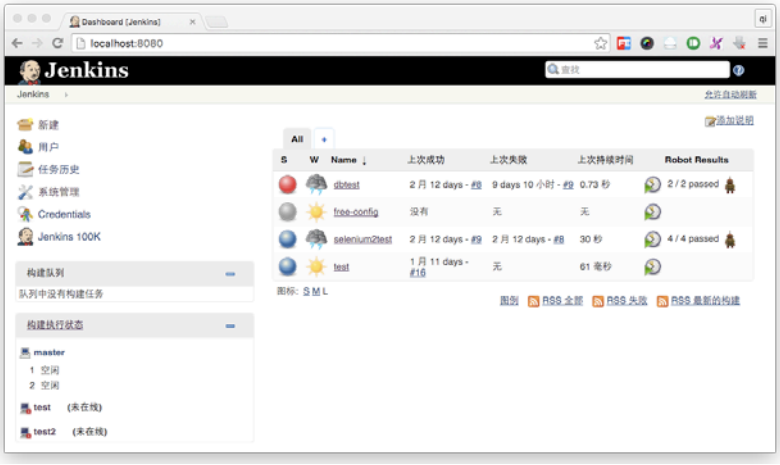

图 8-2-11

同时在配置了 RF 插件的 Job 的首页，会显示构建结果的趋势图，左侧也会多出 Robot Results 以便于查看测试报告，如图 8-2-12 所示。

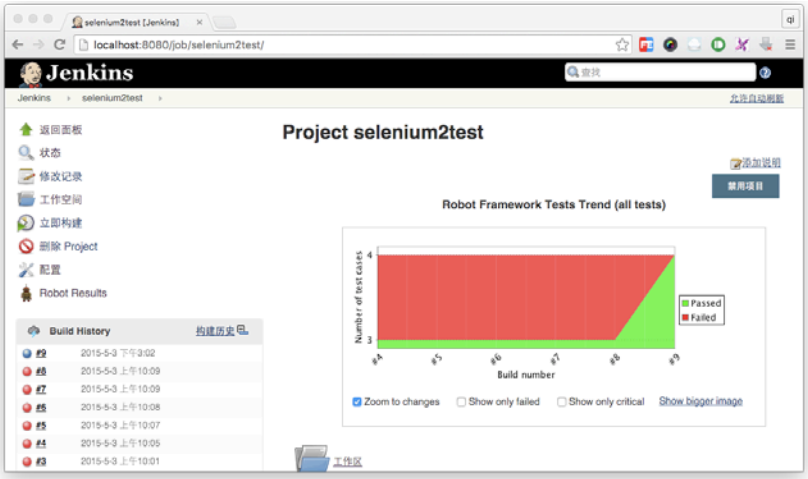

图 8-2-12

8.2.5　多配置 Job

前面介绍的都是自由风格的 Job 配置，这里介绍一下多配置 Job 的差异点，如图 8-2-13 所示。

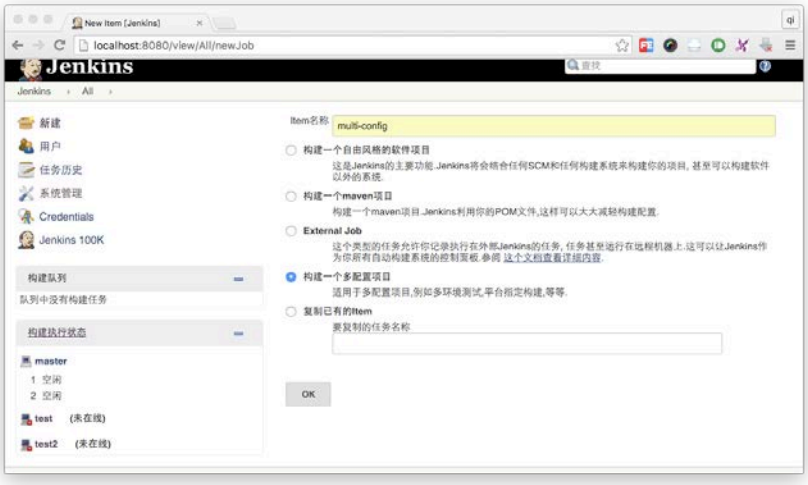

图 8-2-13

1. 配置不同

首先是指定运行 slave 的 Restrict where this project can be run 没有了，取而代之的是 Configuration Matrix，如图 8-2-14 所示。

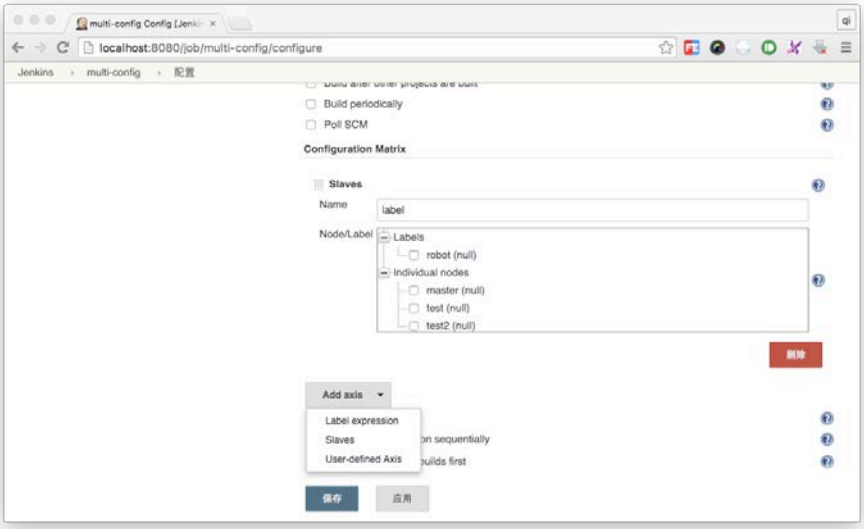

图 8-2-14

单击“Add axis”下拉按钮，会弹出几个选项，通常前 2 个就够用了。

第一个 Label expression 是输入 Label 的表达式，和自由风格的 Restrict 填写是一样的。

第二个 Slaves 是把所有 slave 按照 Labels 和 Individual nodes 分别列出来，直接勾选就可以了。

Labels 就是之前建立节点时添加的标签，如果多个 slave 都有同样的标签，那么它们就是一个 group 了。

这种比较适合有多个 Job 在多台 slave 上执行的情况，因为如果指定具体的一个 slave，会出现抢占资源的情况。而指定一个 group 的话，那么资源会按空闲进行合理分配。

例如，某部门内的十几台机器就是这样来分配的，每个测试组的机器都有单独的标签，整体上所有的机器也有统一的标签，这样在后续调用的时候可以根据情况合理分配。

Individual nodes 其实就是实际的节点名称，直接选择。

2. 显示执行结果不同

多配置项目和自由风格的差别，除了在配置选 slave 标签的差别，另外就是显示执行结果上的差别了。

由于多配置需要有多台 node 执行机来演示，笔者手上只有本机一台机器，所以无法很好地演示这个效果，只能说明一下具体的差异点了。

（1）在 Jenkins 首页可以看到，multi-config 项目的 Robot Results 是空的，没有 RF 结果，如图 8-2-15 所示。

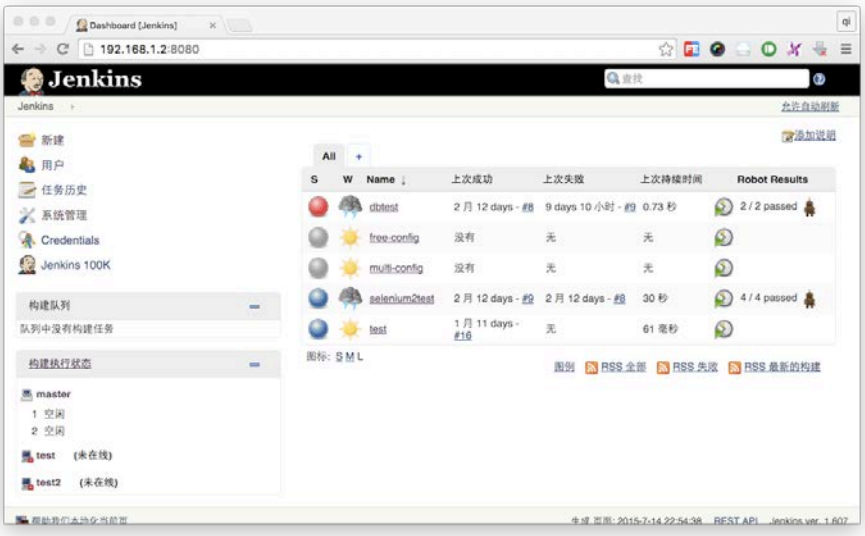

图 8-2-15

（2）在 Job 的首页也没有 RF 的结果，如图 8-2-16 所示。

multi-config 项目设置了多个标签，那么在 Job 首页就能看到每个配置的链接。只有选择这个配置进去后，才能看到实际的结果。

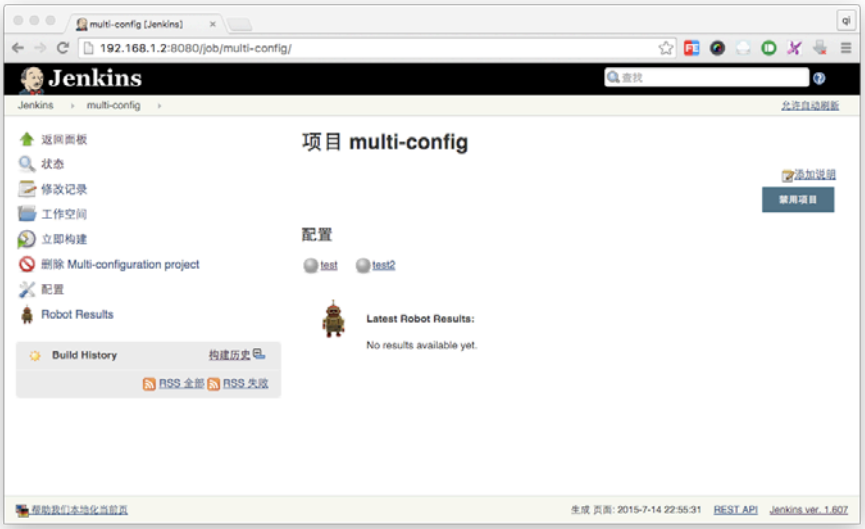

图 8-2-16

8.3 小结

实际上关于 Jenkins 还有很多没有讲到，如果要放开了说内容非常多，多到可以单独再出一本书了。所以这里讲到的主要都是和我们自动化测试 Job 直接相关的，其余那些并不是不重要，而是和本书内容关联不太大，有需要的话可以自行在网上搜索一下。

最后补充 3 点：

1. Jenkins 权限控制。可以和公司的账号体系结合起来，也可以单独用 Jenkins 自己的用户管理。唯独有一点需要切记，在启用安全设置时，授权策略里面，一定要把自己的管理员用户先加到权限最大，否则就只能哭着重装 Jenkins 了；

2. Jenkins 的 URL 配置，最好按 IP 地址或者域名配置，这样方便其他人访问；

3. Jenkins 的邮件配置，最好把发邮件的功能配置好，Jenkins 自带的邮件无法自定义格式，可以考虑使用插件 Editable Email Notification，这个插件可以自定义邮件的内容格式。

9

第 9 章
移动自动化测试

本章主要是介绍如何用 Robot Framework 结合 Appium 进行移动自动化测试，并分别介绍了 Android 和 iOS 的测试案例。

本章的测试案例：

https://github.com/qitaos/rf-demos/tree/master/AppiumLibrary-Demo

9.1 Appium 介绍

Appium 是 Sauce Labs 出品的一个开源的自动化测试框架，可以用来测试 iOS 和 Android 平台上的原生应用、移动 Web 应用和混合应用。原生应用是指使用 iOS 和 Android 的 SDK 开发出来的 App；移动 Web 应用指的是使用移动浏览器访问的应用，比如 Appium 支持 iOS 上的 Safari 和 Android 上的 Chrome，或内置浏览器；混合应用指的是通过 WebView

使得原生代码内容和 Web 内容可以进行交互的 App。

当然，最重要的是 Appium 是一个跨平台的框架，它允许你在 iOS 和 Android 这两个不同平台上使用同样的 API 来编写自动化测试脚本，这样提高了测试代码的复用性。

前面介绍 Web 测试底层用的是 Selenium 框架，其实 Appium 这里也是封装了 Selenium 的框架，其实就可以把移动 App 应用看作是一个网页，然后通过 Appium 进行测试。

Appium 结合 Robot Framework 来做测试，需要先安装好 Appium。

有关 Appium 的安装，如果你使用的是 Mac 平台，可以参考如下网址中的文章的介绍：

https://testerhome.com/topics/1225

如果你使用的是 Windows 平台，可以参考如下网址中的文章的介绍：

https://testerhome.com/topics/2376

这里就不详细介绍了。

安装好 Appium 之后，需要再安装两个 Python 的测试库，一个是 Appium-Python-Client，另一个是 robotframework-appiumlibrary。前者是用 Python 脚本来驱动 Appium 的，后者是基于 Robot Framework 实现的 Appium 库，用到了前者实现的一些基础框架。这两个就类似于 Selenium 和 Selenium2Library 一样。

相关下载地址：

https://pypi.python.org/pypi/Appium-Python-Client/

https://pypi.python.org/pypi/robotframework-appiumlibrary

推荐使用 pip 安装。如果想要手动安装，那就需要查看一下这个 Library 依赖哪些 Library。查看方法是在下载地址把源码包下载下来，解压缩，通常会有一个目录是“.egg-info”结尾的，它有可能在根目录，也有可能在 src 里面，前面的名字一般就是测试库的名字了，比如 Appium_Python_Client.egg-info。找到这个目录后，目录下面会有一个 requires.txt 的文件，打开它就可以看到它依赖哪些 Library 及其版本了。

比如 Appium-Python-Client 依赖的是：

```
selenium>=2.41.0
enum34
```

而 robotframework-appiumlibrary 依赖的是：

```
decorator >= 3.3.2
robotframework >= 2.6.0, <= 2.8.7
docutils >= 0.8.1
Appium-Python-Client >= 0.5
mock >= 1.0.1
sauceclient >= 0.1.0
pytest-cov >= 1.8.1
pytest-xdist >= 1.11
pytest-pythonpath >= 0.4
```

当然，可能其中某个 Library 又会依赖其他 Library，这就要具体地一个个看了，所以手动安装是比较麻烦的事情，如果依赖比较少还可以，依赖多的时候就不如 pip 省心了。

在 Appium 安装好之后，需要运行一下 appium-doctor 来检查一下环境是否都配置好了。可以通过命令行运行，也可以通过单击 Appium 客户端界面上的第 3 个按钮（比较像医生听诊器的那个），Mac 版客户端界面，如图 9-1-1 所示。

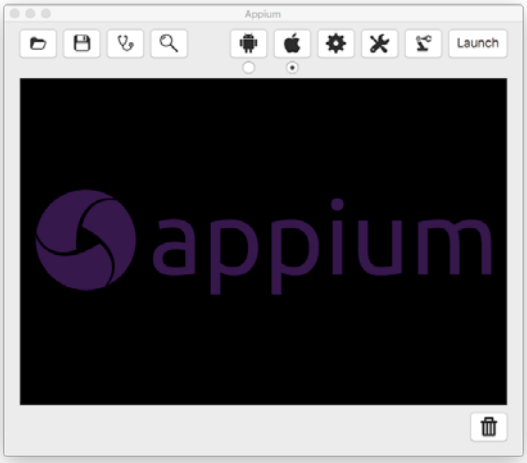

图 9-1-1

单击此按钮后，如果看到类似于下面的信息，那就是配置正确了。如果有任何异常，会提示你要修复异常，比如缺少某些安装包或者没有配置环境变量等。

/Users/qitao >'/Applications/Appium.app/Contents/Resources/node/bin/node' '/Applications/Appium.app/Contents/Resources/node_modules/appium/bin/appium-doctor.js'

Running iOS Checks

✔ Xcode is installed at /Applications/Xcode-beta.app/Contents/Developer

✔ Xcode Command Line Tools are installed.

✔ DevToolsSecurity is enabled.

✔ The Authorization DB is set up properly.

✔ Node binary found at /usr/local/bin/node

✔ iOS Checks were successful.

Running Android Checks

✔ ANDROID_HOME is set to "/Users/qitao/Desktop/mobile/android/adt-bundle-mac-x86_64-20140321/sdk"

✔ JAVA_HOME is set to "/Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home."

✔ ADB exists at /Users/qitao/Desktop/mobile/android/adt-bundle-mac-x86_64-20140321/sdk/platform-tools/adb

✔ Android exists at /Users/qitao/Desktop/mobile/android/adt-bundle-mac-x86_64-20140321/sdk/tools/android

✔ Emulator exists at /Users/qitao/Desktop/mobile/android/adt-bundle-mac-x86_64-20140321/sdk/tools/emulator

✔ Android Checks were successful.

✔ All Checks were successful

都安装好后，我们就直接用例子入手了。这里使用官方示例的安装包进行测试，Android 的基本可以直接用，iOS 的如果是真机，需要大家有开发者证书才能打包调试，否则就只能用模拟器。不过据说最新的 xcode7 不要证书就可以调试了，现在 xcode7 beta 版本已经可以在开发者中心下载使用了，估计大家见到本书的时，xcode7 正式版应该已经出来了，这样应该就不需要证书也可以直接来运行案例了。

在运行案例之前，需要有一个动作来启动 Appium，无论是命令行或者客户端，先要把 Appium Server 启动起来，监听本地服务器的是 4723 端口，所以启动 Appium Server 后会看到如下信息：

```
Appium REST http interface listener started on 0.0.0.0:4723
```

然后才能通过“http://localhost:4723/wd/hub”去触发相应的操作。在本章的例子中，笔者是使用客户端来启动的，客户端界面，如图 9-1-2 所示。

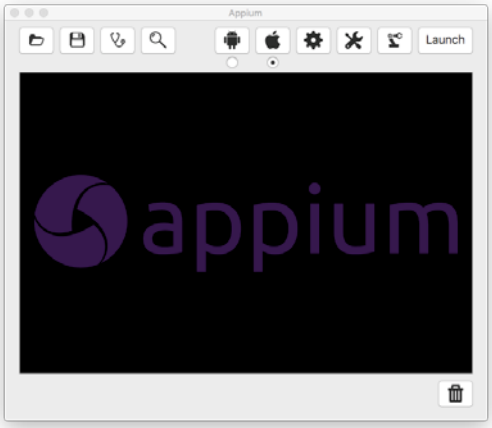

图 9-1-2

要选择 Android 或 iOS 的图标，然后单击“Launch”按钮来启动 Server。

有一个比较麻烦的地方是，无论案例失败还是执行完成，要进行下一次案例的运行之前，都需要先停止 Server，再重新启动 Server，否则案例是无法运行的。所以大家熟悉之后，可以试着结合前面的一些内容，尝试使用命令行来触发“Launch Server”和“Stop Server”，界面如图 9-1-3 所示。

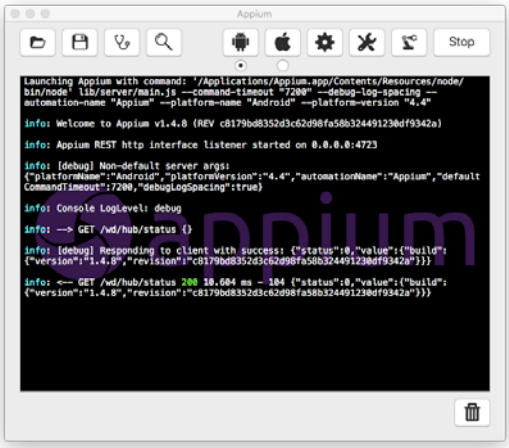

图 9-1-3

9.2 Android 自动化测试

9.2.1 模拟器安装

可以选择连真机还是连模拟器，模拟器的话推荐使用 Genymotion，官方网站是：

https://www.genymotion.com/

Genymotion 号称最快的 Android 模拟器，实际体验来说确实比 Android 自带的模拟器快很多，只要免费注册一下就可以下载了。Genymotion 网站页面，如图 9-2-1 所示。

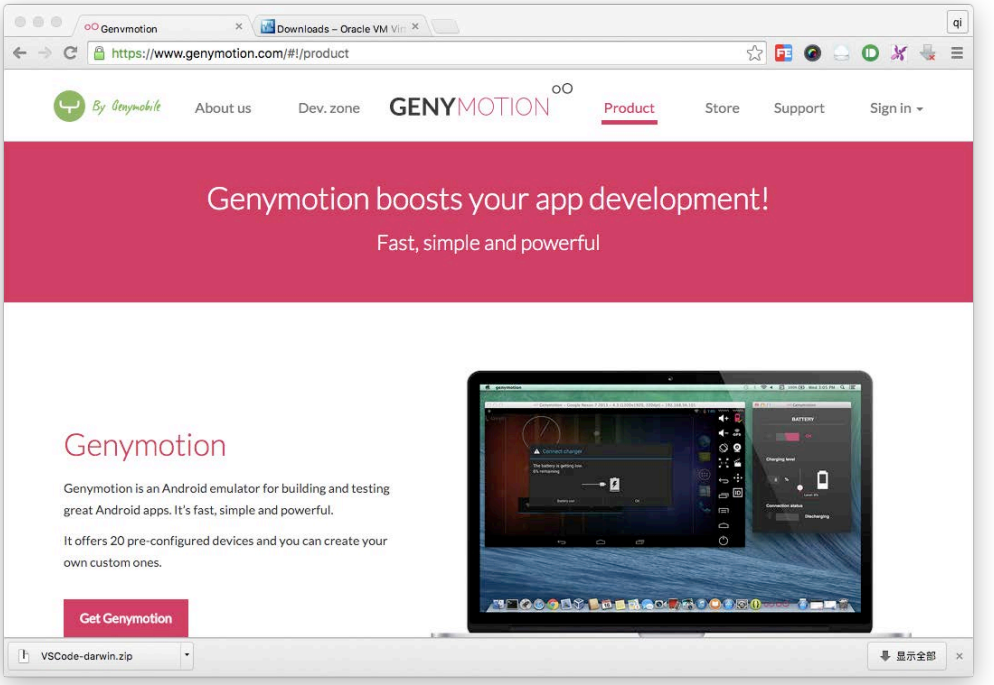

图 9-2-1

要使用 Genymotion，还需要安装 Oracle VirtualBox，官方网站是：

https://www. virtualbox.org/wiki/Downloads

除了下载主程序安装，还要下载 Oracle VirtualBox Extension Pack，在官网下载页面上都有。VirtualBox 网站页面，如图 9-2-2 所示。

图 9-2-2

都安装好之后就可以打开 Genymotion 了，界面如图 9-2-3 所示。

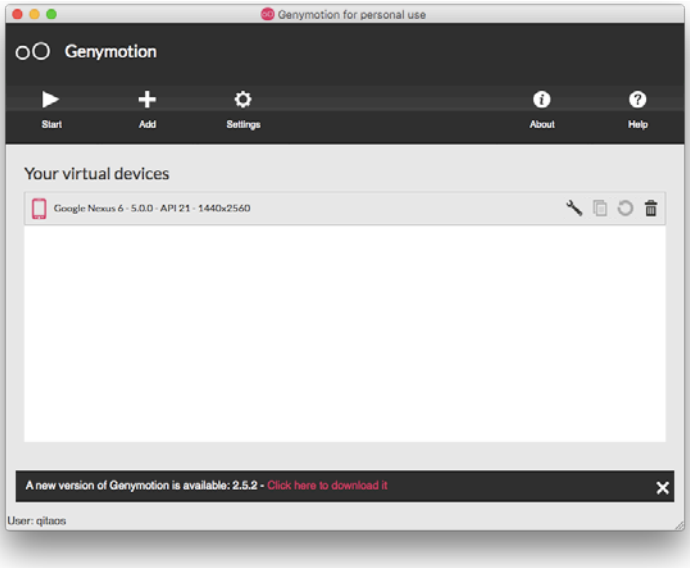

图 9-2-3

如果你没有添加过设备，请单击“Add”按钮去添加个设备，期间可能会要求你登录。根据自己的需要选择设备，然后“一路”单击“Next”按钮，等待它下载完成就可以了。

已经添加完设备的朋友，可以选择模拟器，然后单击“Start”按钮启动模拟器。启动过程，如图 9-2-4 所示。

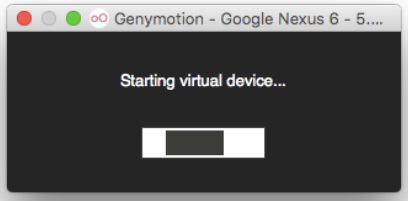

图 9-2-4

加载成功后就可以看到模拟器的界面了，如图 9-2-5 所示。

图 9-2-5

运行之前记得要把模拟器解锁一下。

9.2.2　测试案例

接下来看看测试案例，Appium 官方提供的 sample-code 里有基于 Python 或其他语言的测试代码，也有基于 Robot Framework 的测试案例，Robot Framework 的测试案例只提供了 Android 的测试案例。sample-code 里 Android 的 APK 是 ContactManager，一个联系人管理的应用。官方这个案例笔者试了一下，无法直接运行，主要问题是几个文本框在案例中是有 id 的，但是在 uiautomatorviewer 中却查看不到 id，于是笔者稍微调整了一下脚本，用 xpath 来代替，终于能跑了。查看不到 id 的是用的 Google Nexus7 的机型，后来换了个三星的 Note3 就可以查看到 id，看来还是和机型有关系。所以案例准备了两份，一个是给没有 id 的，一个是给有 id 的，差异只有文本框输入那里的差别。

不过仍然存在一个问题，就是在用模拟器打开这个应用，添加联系人，到最后一步提交的时候都会出现 crash，试了 4.4 和 5.0 的模拟器都会这样。虽然有个 bug，不过这里的案例能够展示给大家如何写测试案例就可以了。运行案例之前一定要先启动 Appium Server。测试脚本见表 9-2-1。

表 9-2-1

Open Application	http://localhost:4723/wd/hub	platformName=Android	platformVersion=4.2.2
...	deviceName=192.168.56.101:5555	app=${CURDIR}${/}app${/}ContactManager.apk	appPackage=com.example.android.contactmanager
Click Element	accessibility_id=Add Contact		
${source}	Get Source		
Log	${source}		
Sleep	5s		
Input Text	xpath=//android.widget.TableRow[contains(@index,3)]/android.widget.EditText	Jacky Qi	
Input Text	xpath=//android.widget.TableRow[contains(@index,5)]/android.widget.EditText	13800138000	
Input Text	xpath=//android.widget.TableRow[contains(@index,7)]/android.widget.EditText	qitaos@gmail.com	
Click Element	accessibility_id=Save	accessibility	

前两行其实是一行代码，笔者使用“...”来做了换行处理，是为了让大家看得更清楚，如果写成一行的话，请去掉“...”就行了。这个 Open Application 是启动程序的关键字，第一个参数 http://localhost:4723/wd/hub 是 remote_url，是用于通过 Appium Server 触发操作的接口，platformName=Android 告诉 Appium 这是 Android 平台的应用。platformVersion=4.2.2 是告诉它 Android 的最低版本，因为 Appium 驱动 Android 应用的时候，如果 Android 版本低于 4.1 是使用 Seledroid 来驱动的，如果版本是高于 4.1 是使用 UIAutomator 驱动的。deviceName=192.168.56.101:5555 是设备名称，可以用 IP 端口，也可以用 deviceid，这里的 IP 端口是模拟器默认使用的，如果是真机可以用 deviceid。“app”是安装包的路径，appPackage 是安装包的包名。如果用自己的安装包，请自行更改。

启动应用进入首页，如图 9-2-6 所示。

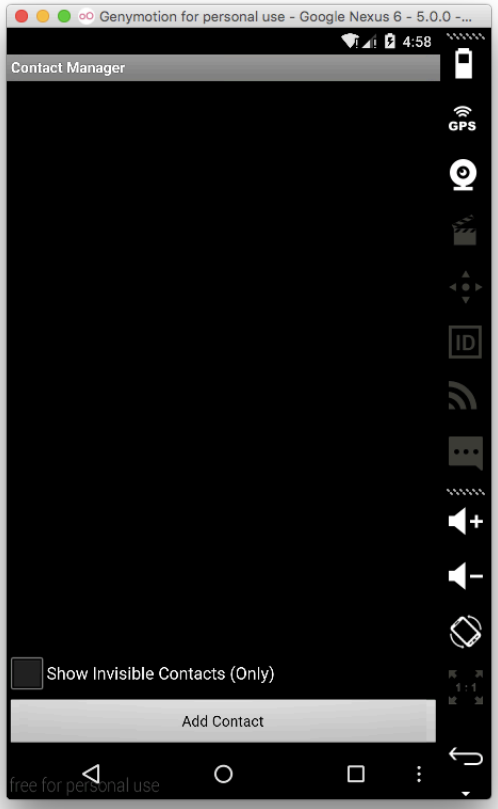

图 9-2-6

然后在界面第一行的选项中选择“Add Contact”选项，然后进入录入联系人信息的页面，如图 9-2-7 所示。

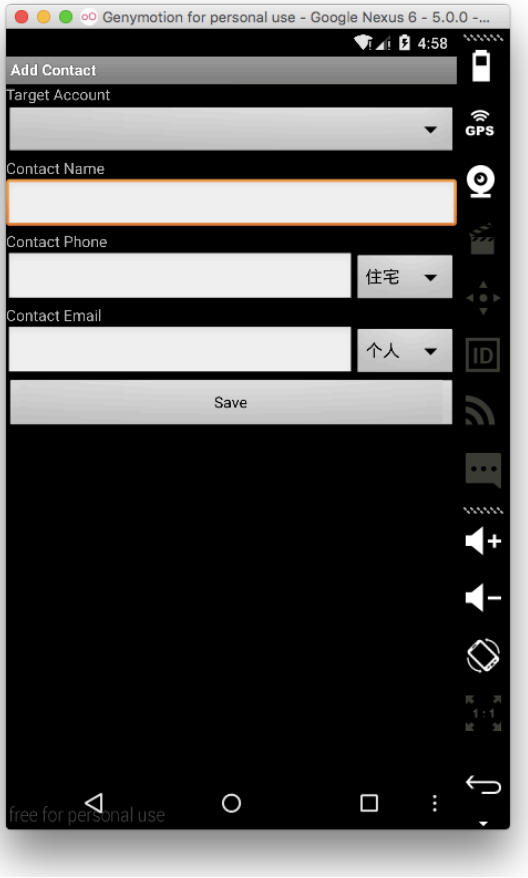

图 9-2-7

依次录入姓名、电话、邮件，然后单击“Save”按钮，如图 9-2-8 所示。

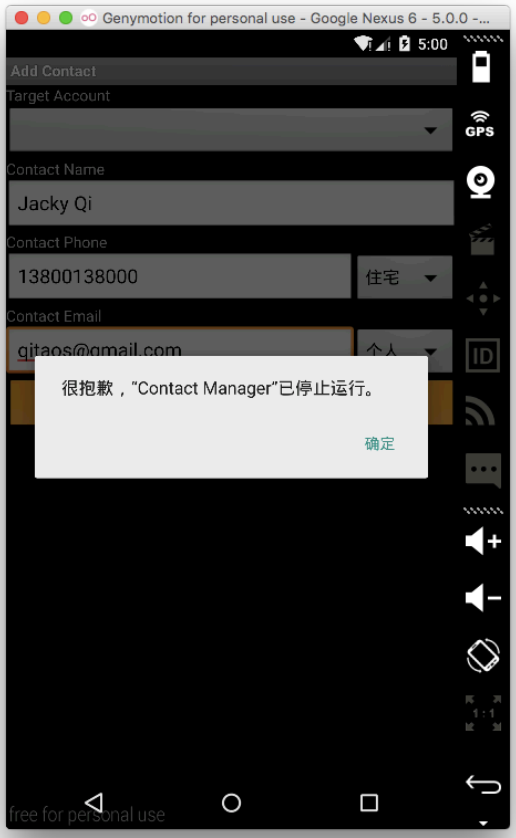

图 9-2-8

虽然应用“挂了”，不过前面的步骤，就是案例中的步骤，也执行完成了，因为没有加检查点，所以案例结果是 Pass。如果加上检查点就可以发现异常，进行截图了。

9.2.3 Android 对象识别

在 Web 页面上，有很多工具可以查看到页面元素的属性，以便于写案例时使用，在移动端也有这样的工具。在 Android 平台就有自带的 UI Automator Viewer，它位于 android-sdk 的 tools 目录下，进入该目录，找到 UI Automator Viewer。启动它之后，要单击左上角菜单栏的第二个像手机一样的图标，此时稍微等一会，就会把当前模拟器（真机也一样）里的界面展示出来，在左侧单击某个对象，右侧上半部分会显示它的布局结构层级，下半部分就是这个对象的一些属性值。比如这里的 Contact Name，在界面右边就找到

了它的 resource-id，这是在三星 Note3 机型上抓的，所以在有 id 的案例里可以用这个 id 来执行输入文本框的操作。而在 Nexus 机型上，UI Automator Viewer 抓出来的属性没有 resource-id，原因不太清楚，但就是因为它没有这一项，所以在用 resource-id 执行输入文本框操作的时候就报错了，因而笔者才将案例换成了 xpath 操作。

UI Automator Viewer 抓取界面不是实时的，如果此时你的模拟器上因为你的操作换了页面或者页面出现一些变化，是不会实时同步到 Viewer 的，你需要再次单击左上角菜单栏第二个像手机的图标，才会获得最新的页面信息。UI Automator Viewer 界面，如图 9-2-9 所示。

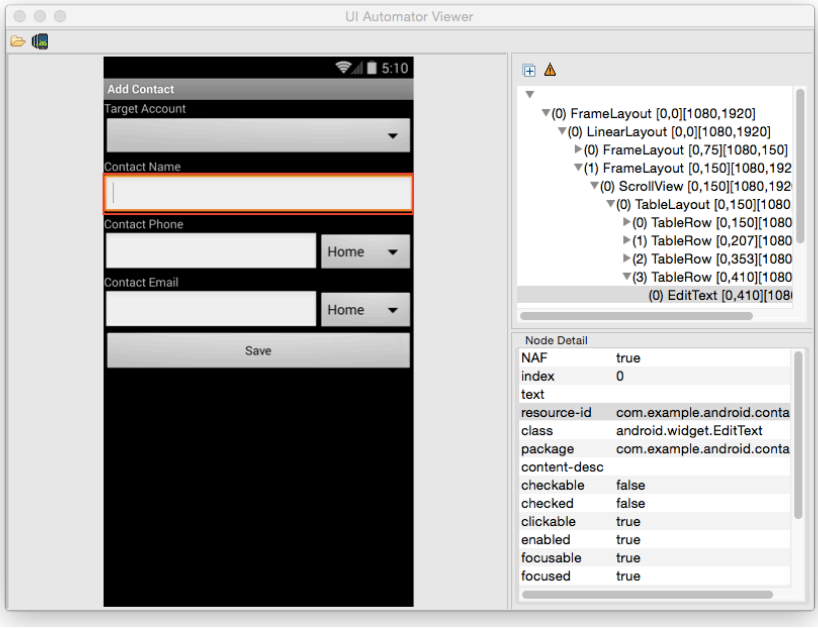

图 9-2-9

9.3　iOS 自动化测试

9.3.1　测试案例

正如本章开始说的，在目前 xcode6 来说，打包需要个人证书，如果没有的话只能用模拟器，无法用真机的。如果 xcode7 及时发布，应该对大家来说是个福音，不用专门去

购买个人证书了，当然如果要提交 App Store 还是需要个人证书的。

要运行 iOS 的例子，那么首先要有个 Mac，因为 Android 自动化可以在 Windows 或 Mac 上运行，但是 iOS 的就只能在 Mac 上运行了。所以要同时“玩”Android 和 iOS，最方便的就是统一在 Mac 上“玩”了。当然买个 Mac 的成本要高一些，不过从使用寿命、便携性、电量等角度来说，买个 Mac 要划算很多。

那么假定已经有 Mac 了，对于 iOS 的例子，这里用模拟器来操作，模拟器只要装了 xcode 都会有，不过历史版本的模拟器就需要手动下载安装了。这里比较好的一点是不用像 Android 一样先启动模拟器，Appium 会自动帮你启动模拟器的。但是同样都需要先把 Appium Server 先启动起来。先看看案例，见表 9-3-1。

表 9-3-1

Open Application	http://localhost:4723/wd/hub	alias=Myapp1	platformName=iOS	platformVersion=9.0
...	deviceName=iPhone 6	app=${CURDIR}${/}app${/}TestApp.app		
Input Text	name=TextField1	13		
Input Text	name=TextField2	22		
Click Button	Compute Sum			
Click Button	done			
Click Button	show alert			
Click Button	OK			
Swipe	178	356	133	356

这里和 Android 一样，Open Application 是启动应用程序的关键字，参数基本上和 Android 类似，第一个参数 http://localhost:4723/wd/hub 是 remote_url，是用于通过 Appium Server 触发操作的接口，alias=Myapp1 是给当前的应用定义一个别名，如果启动多个应用可以通过别名进行切换。platformName=iOS 告诉 Appium 当前是 iOS 平台的应用。platformVersion=9.0，这个就需要和你本机的模拟器 iOS-SDK 版本匹配了，deviceName=iPhone 6 是 iOS 的设备名称，不需要写模拟器，Appium 会根据你的 iOS-SDK 版本拼出模拟器的设备名称，如果是真机也会直接使用真机的设备。这里用的是 9.0 的版本，笔者现在属于提前“尝鲜”，用的 xcode7 beta 版本，iOS 9.0 beta 版本，由于此时的 Appium 还没有兼容 xcode7，因此笔者尝试修改了它的个别源码，才能正常运行。本书出版的时应该能看到正式版发布了，相信那时 Appium 也会做兼容适配，不需要改源码了。

“app”是我们用 xcode 打包编译后的 app 文件，也可以用 ipa 文件。这里笔者是用官方的工程重新打了个新的 app 文件，大家也可以用自己的工程打包文件替换。

成功启动应用后就会看到主界面了，如图 9-3-1 所示。

接着案例会在两个文本框分别输入 13 和 22，然后单击“Compute Sum”选项来计算结果，如图 9-3-2 所示。

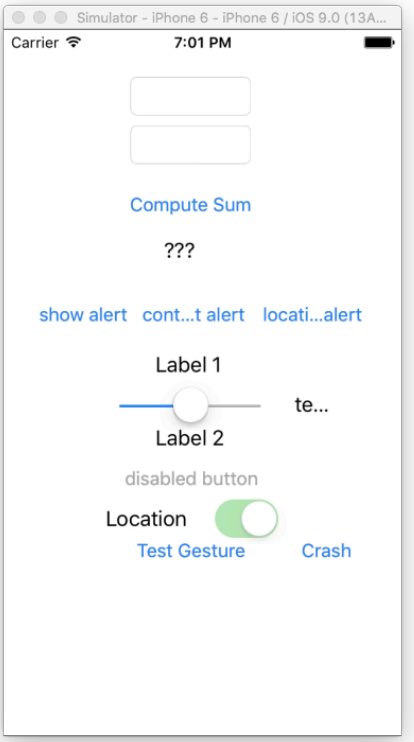

图 9-3-1

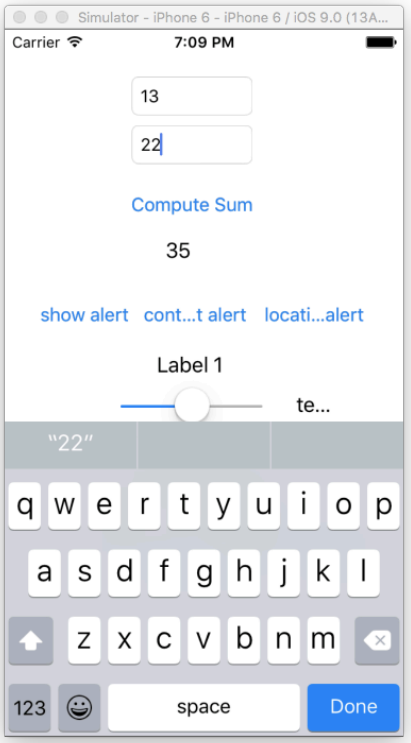

图 9-3-2

然后单击“done”是关掉键盘，至于为什么不是“Done”，等会后面介绍了如何查看对象，大家自己去看。然后单击“show alert”选项后弹出个对话框，如图 9-3-3 所示。

单击“OK”按钮之后，用 Swipe 滑动了一些，如图 9-3-4。

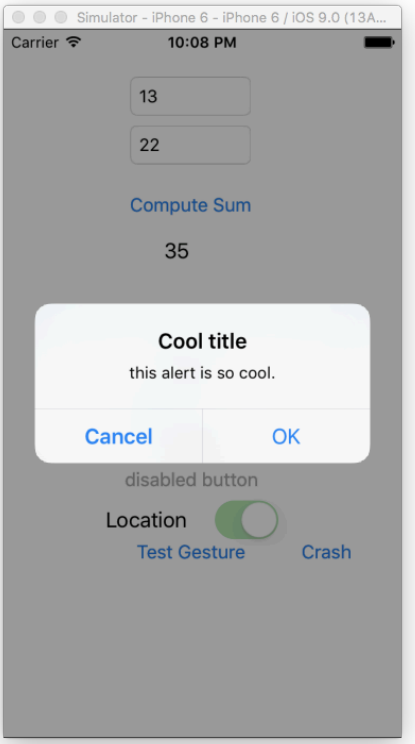

图 9-3-3

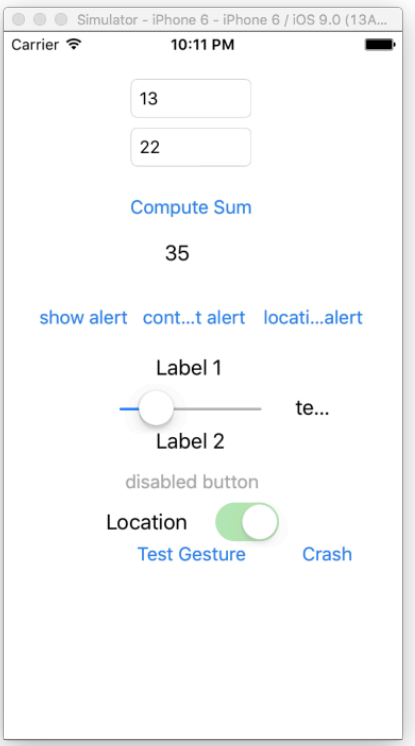

图 9-3-4

Swipe 是滑动的意思，它的 4 个参数是滑动的起点和终点的坐标，其实还有第 5 个参数是一个时间，如果你想慢滑，可以把时间加长点。一开始这里出现滑不动的情况，笔者在前面加了个 Long Press 长按，不过仔细看了一下 Swipe 的源码，发现它其实是先 press（按住）起点坐标，然后 wait（等待）设定的时间，然后 move（移动）到终点坐标，最后 release（放开）按住的动作。后来注释掉了 Long Press，发现又可以滑动了。

9.3.2 iOS 对象识别

iOS 识别比 Android 更方便些，Appium 首页，如图 9-3-4 所示。在 Appium 首页上从左往右数的第 4 个“放大镜”按钮，就是 Appium Inspector，用来查看页面对象的。据说 Android 和 iOS 都可以用，不过这里主要用来查看 iOS 的，Android 的用前面介绍的 UI Automator Viewer 就行了。

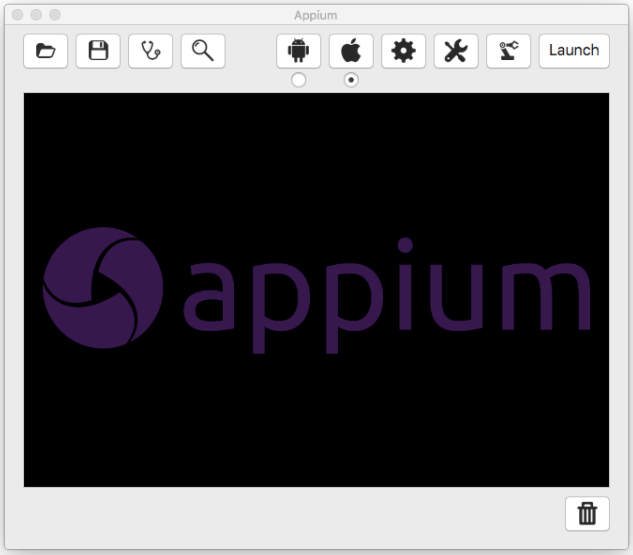

图 9-3-4

如果你想先查看对象，可以单击“苹果”图标，勾选“App Path”复选框，选择或者输入安装包的路径，勾选“Force Device”复选框，指定用什么设备执行，如图 9-3-5 所示。

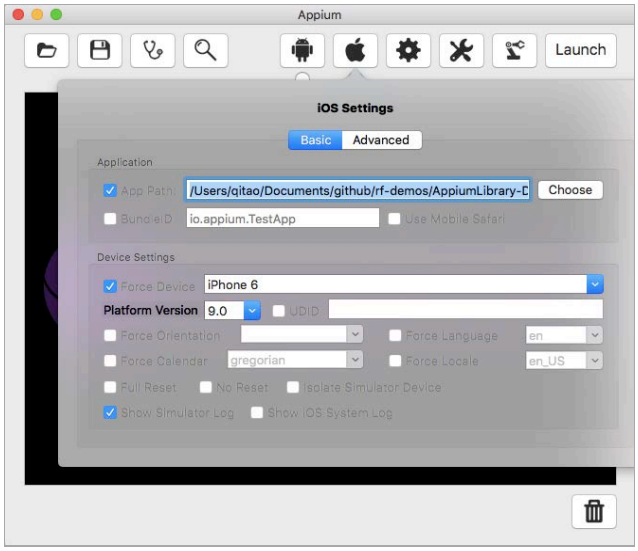

图 9-3-5

然后单击“锯齿”图标，界面如图 9-3-6 所示。勾选“Prelaunch Application”复选框，这样在启动 Appium Server 的时就会自动启动模拟器安装应用。这个只是为了方便我们用 Appium Inspector 来查看对象的，执行测试案例时就不要勾选这个复选框了。

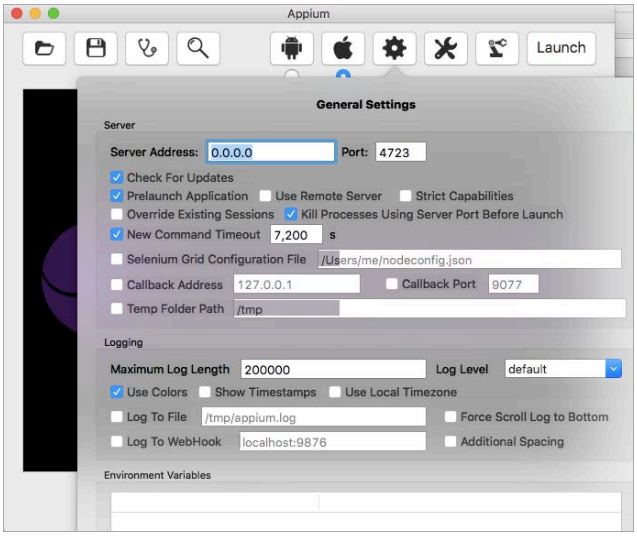

图 9-3-6

当模拟器运行之后，进入到应用界面，此时单击 Appium 主界面上的“放大镜”按钮，就会打开“Appium Inspector”界面，如图 9-3-7 所示。

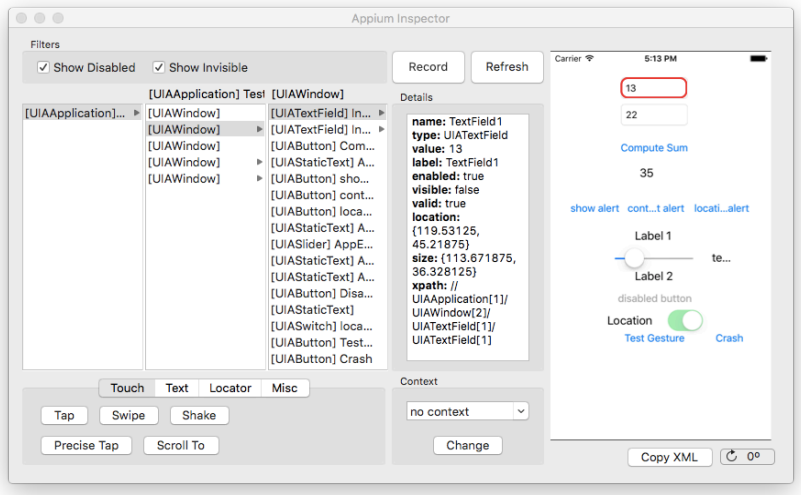

图 9-3-7

和 Android 类似，可以单击右边的具体某个元素，左边就会展示出它的布局结构和属性。如果页面发生变化了，请单击“Refresh”按钮。如果想录制脚本，可以单击“Record”按钮，这个录制和 xcode 里的 instruments 是一样的。这里最方便的是直接提供了 xpath，如果对象没有比较方便操作的属性，可以直接用 xpath 来操作。

9.4　小结

本章初步介绍了一下如何将 Appium 结合 Robot Framework 进行自动化测试，并且提供了比较简单的例子。如果要在比较复杂的应用里做自动化测试案例来测试该应用，还需要多多尝试。毕竟 Appium 现在是 1.4.8 版本，还需要进一步摸索。已经知道的情况，比如每次启动 Appium Server 和停止 Appium Server，这里可以考虑用 OperatingSystem 测试库里的 run 来以命令行的方式来启动和停止。在 iOS 的案例执行时会比较慢，没有具体看是 AppiumLibrary 慢还是 Appium 慢，原因待查。可能还有一些不知道的情况，期待大家发现。

第三部分　大乘篇

- 第 10 章　自定义你的 RF

10

第 10 章

自定义你的 RF

现在就要进入到深入学习了，建议学习的人最好有一定的 Python 基础。本章主要介绍了如何修改测试库、如何自己重新创建一个测试库。

本章的例子：

https://github.com/qitaos/rf-demos/tree/master/TestLibrary/ExampleLibrary

10.1 修改 Selenium2Library 测试库

其实在前面介绍 Selenium2Library 的时候提到了，对 Selenium2Library 做了一些修改和扩充，分别看一下修改和扩充的例子。

首先要想修改或者扩充 Library，首先要知道 Library 安装后的路径，所有的 Library 安装后都会在一个叫做 site-packages 的目录里，找到这个目录就找到 Library 了，在

Windows 平台最好找了，它就在 python 安装目录下的 lib 目录里，在 Mac 平台上通常是位于这个路径的：/Library/Python/2.7/site-packages。

知道了路径，就可以找到对应的文件并进行修改了。先说说修改，Selenium2Library 的截图方法其实有一个问题，就是截图是基于浏览器来截图的，如果浏览器出问题异常关闭了，此时就截不了图了。当然，有人说浏览器都没有了，截图也没什么意义了。不过有些情况浏览器没有了，有可能会有其他的报错信息在，这时候截图就有意义了。因此就来改造截图。先看一下原来的截图方法，位于 Selenium2Library 目录的 keywords 目录里的 _screenshot.py 文件里，主要的截图方法如下：

```
path, link = self._get_screenshot_paths(filename)
if hasattr(self._current_browser(),'get_screenshot_as_file'):
    self._current_browser().get_screenshot_as_file(path)
else:
    self._current_browser().save_screenshot(path)
self._html('</td></tr><tr><td colspan="3"><a href="%s">'
    '<img src="%s" width="800px"></a>' % (link, link))
```

可以看到这里直接使用了“self._current_browser()”，在某些异常导致浏览器关闭的情况下，这里就会直接报错了。

这里修改其实比较简单，因为 RF 本身有自带一个 Screenshot 截图的 Library，要做的就是把这个库的方法引用过来，虽然这导致了这个库要依赖 Roboto Framework，当然肯定会有 Robot Framework，所以这种依赖也没有什么问题。首先在_screenshot.py 的上面增加 Screenshot 库的引用：

```
from robot.libraries.Screenshot import Screenshot
```

然后注释掉上面笔者贴出来的主要内容，换成如下脚本：

```
if not filename:
    filename = "selenium-screenshot"

path = Screenshot().take_screenshot_without_embedding(filename)
self._html('</td></tr><tr><td colspan="3"><a href="%s">'
    '<img src="%s" width="800px"></a>' % (path, path))
return path
```

这里笔者选择了 Screenshot 里的 take_screenshot_without_embedding 方法，参数是 filename，而 filename 笔者稍稍处理了一下，给了个固定名字。如果想改回原来的方式也可以尝试自己修改一下。随后 self._html 方法里把原来的 link 换成 path。后面笔者额外加

了一个 return path，这样需要得到这个路径的朋友，可以自己通过关键字的返回值获取到。

接着说说扩充方法，笔者扩充了两个方法，get_table_rows 和 get_table_cols_at_row。前面的方法的作用是获取当前表格的行数，后面的方法是获取某一行的表格有多少列。为什么会增加这两个方法呢？因为其实笔者以前一直用的是 QTP，QTP 的 webTable 对象就有这样两个方法。

在 Selenium2Library 里对于表格的操作仅仅有一个 get_table_cell 的方法，其他的都是一些断言类型的方法。从实际使用来说是不够的，因为 get_table_cell 方法需要你清楚地知道要获取的单元格内容在表格的具体哪一行哪一列，而实际上做测试的时候只能知道表格里会有这条数据，列有可能能明确知道，但是并不知道是哪一行。就算想做个循环来遍历一下所有的行，那也需要知道总行数才能去遍历。反过来可能知道数据在某一行，但是不知道在哪一列，此时如果不知道这一行的列数也是无法去遍历的。

由此笔者才考虑实现这两个方法来满足测试的需要，在实现这两个方法的时候，其实基本上都参考了 get_table_cell 里的一些写法和用法。首先看一下 get_table_rows 方法：

```
    def get_table_rows(self, table_locator, loglevel='INFO'):
            """add by qitao016. 20131115
            get_table_rows: return the rows of the table.
            """
    table = self._table_element_finder.find(self._current_browser(),
table_locator)
    if table is not None:
        rows = table.find_elements_by_xpath("./thead/tr")
        rows.extend(table.find_elements_by_xpath("./tbody/tr"))
        rows.extend(table.find_elements_by_xpath("./tfoot/tr"))
        return len(rows)
    self.log_source(loglevel)
    raise AssertionError("Table %s could not be found." % table_locator)
```

上面就是扩展的一个方法的脚本，添加在 keywords 目录下的_tableelement.py 文件里。

实际上这个方法很简单，基本上“脱胎”于 get_table_cell 方法。首先是根据 table_locator 来定位这个 table，如果找到了 table 对象，即 table is not None。则在 table 里根据“./thead/tr”查找行对象，然后依次扩展“./tbody/tr”、“./tfoot/tr”，这样是为了兼容各种有 thead 和 tfoot 的 table。当然从设计上或许还可以有更优化的写法，比如只要一行就可以代替那 3 行了：

```
    rows = table.find_elements_by_xpath(".//tr")
```

不过笔者还是保留了最初的写法，这样和原版看起来风格比较一致。

接着看一下另一个方法 get_table_cols_at_row：

```
   def get_table_cols_at_row(self, table_locator, row, loglevel='INFO'):
          """add by qitao016. 20131115
          get_table_cols_at_row: return the cols of the table in one row.
          """
          row = int(row)
          row_index = row - 1
          table = self._table_element_finder.find(self._current_browser(),
table_locator)
          if table is not None:
             rows = table.find_elements_by_xpath("./thead/tr")
             if row_index >= len(rows): rows.extend(table.find_elements_
by_xpath("./tbody/tr"))
             if row_index >= len(rows): rows.extend(table.find_elements_
by_xpath("./tfoot/tr"))
             if row_index < len(rows):
                columns = rows[row_index].find_elements_by_tag_name('th')
                columns.extend(rows[row_index].find_elements_by_tag_name
('td'))
                return len(columns)
          raise AssertionError("Table %s in row #%s could not be found." %
(table_locator, str(row)))
```

这里面除了 table_locator 参数，多了一个 row 的参数，也就是要指定是在哪一行来获取列数。根据行数，可以得到一个 row_index，行的序列，因为在 python 里一个列表的 index 都是从 0 开始的，所以是 row-1 得到 row_index。然后同样根据 table_locator 定位到这个 table，table 存在的话，先获取“./thead/tr”的行数，这时候增加了两个判断，row_index >= len(rows)，其实就是如果要找的那行的 row_index 大于等于 thead 的行数，说明它不在 thead 的行里，那么就 extend（扩充）“./tbody/tr”的行进来。如果 row_index 还是大，说明不在 tbody 里，那么就要继续扩充“./tfoot/tr”的行进来。

接着判断，如果 row_index 小于总行数了，说明要找的行在范围内，否则说明 row_index 给的超出表格行数了。就好比说一个 3 行的表格，你要找它第 4 行有几列，那是肯定不行的，这样就会去报错了。

row_index 在范围内的话，就要在 rows[row_index]这一行里找所有的列了，首先是 th 的，然后再扩展 td 的，因为 thead 里的 td 有可能会写成 th，所以先找 th 的，再找 td 的，

总之都是在这一行里，所有的列都找出来了，最后把 len(columns)这一行的列数返回。

实际上还有几个吴穹博士帮助扩展的方法，比如获取某个文本在这一行的第几列或者在这一列的第几行等，这里就不具体介绍了，后续笔者也会将扩充的方法提交给 Selenium2Library，争取在后续的版本中更新进去。

在一个 TestLibrary 里修改或扩充方法，首先需要理解这个 Library 的逻辑、构造，明确要扩展的方法归属，大部分在 github 上的开源工程，作者都会把方法分门别类，所以不要到不相干的地方去扩充，例如 Selenium2Library 里不要去 browser 部分去扩充 table 的方法，因为 table 的方法在_tableElement.py 里。如果在错误的地方扩充，有可能只能你自己本地使用了，想要提交给原作者，估计会被打回或者调整。其次，每一个方法的修改或者扩充，都需要经过充分的测试，不要不经过测试就直接给其他人使用，特别是你要想提交给原作者，那都是要写 acceptance test 验收测试案例的。最后，因为整个 Robot Framework 都是基于 Python 的，所以从编程的角度来说，没有什么限制，实现一个功能有很多种方法或设计思路，上面只是给了笔者的设计思路，或许你还有更好的想法，都可以自己去尽情发挥。

10.2 编写测试库

前面都是在原有的 Library 基础上进行修改或扩充，这里会给大家介绍如何做一个新的 Library，当然是基于 Python 的。原本会有一节基于 Java 的，但是由于 Java 的 Library 需要运行在 Jython 下，整个 Robot Framework 这一套都要弄一个基于 Jython 的版本，而在笔者本地调试了很久，一直没有搭建好环境，怎么运行都识别不了 Java 的 Library。因为本书涉及的所有内容，笔者都在本机预先执行调试过很多次，所以本着对大家负责的态度，加上笔者是个 Python“重度患者”，这个部分就不能提供给大家了，或许笔者后续会单独写一本基于 Java 的 Robot Framework 的书籍。

Python 的测试库笔者准备介绍两种模式的，一种是比较简单的单文件的，另一种是稍微结构复杂些的。不过里面不会有太多方法，都是几个比较简单的方法。Library 里的方法都是根据大家的需要自己来完成的，说白了测试库其实就是把一些方法通过 Python 实现，然后提供简单的接口给测试案例使用，所以要加什么方法可以自由发挥，笔者做的例子只是帮助你了解怎样做一个测试库。

10.2.1　测试库分类

实际上从测试库类型上来说分为 Static 静态、Dynamic 动态和 Hybrid 混合这 3 种，作为本书来说，重点介绍 Static 静态类型，其他两种类型需要理解深入一些才能领会，官方的关于 Library 的文档在如下路径的第 4 部分：

http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

10.2.2　测试库结构

通常最容易建立测试库的就是一个单独的文件，只要把这个文件放到 site-packages 目录下就可以生效了。先来看一下测试库的例子。

文件名 ExampleLibrary.py，文件内容如下：

```
#encoding=utf-8

import string
import random
from robot.api import logger
import logging
import sys

class ExampleLibrary:

    ROBOT_LIBRARY_SCOPE = 'GLOBAL'
    ROBOT_LIBRARY_VERSION = 0.1
    ROBOT_LIBRARY_DOC_FORMAT = 'reST'

    def __init__(self, arg = 1):
        """Library 文档 *斜体* 这个文档用的是 reST 结构 reStructuredText__.

           __ http://qitaos.github.io
        """
        print 'Library arg %s' % arg
        pass

    def gen_nums(self, counts):
        """生成随机数字字符串. \n
            例子:
            | ${a}= | gen nums | 4 |
            这样会返回一个随机 4 位数字字符串. 比如 '0624','1456'.
```

```
            """
        s = self._gen_nums(counts)

        print '*INFO* Get random number string: %s' % s
        return s

    def _gen_nums(self, counts):

        li = string.digits
        s = ''
        for n in range(0, int(counts)):
            s += li[random.randint(0, len(li) - 1)]
        return s

    def arg_demo(self, arg1, arg2= 2, *args):
        print 'arg1 %s arg2 %s' % (arg1,arg2)
        for arg in args:
            sys.__stdout__.write('Got arg %s\n' % arg)

    def freearg_demo(self, **freearg):
        for name, value in freearg.items():
            print name, value

class OtherLibrary:

    def __init__(self):
        self._counter = 0
        pass

    def count(self):
        """用于计数器. Init countNum is 0.
        When you call this method it will add 1 by itself.
        Example:
        | ${a}= | count |
        """
        self._counter += 1
        logger.debug('self._counter += 1')
        return self._counter

    def clear_counter(self):
        """clear counter has only a short documentation
        Example:
        | ${a}= | clear counter |
```

```
        """
        self._counter = 0
        logging.info('self._counter = 0')
```

从测试库结构来说，一般分为 head 和 body，实际没有严格意义的区分，都是一个文件里的内容，只是简单区别一下。以上面的代码为例，在 class ExampleLibrary 前面的都可以算做 head 的内容，一般是编码声明，还有就是需要 import 加载的模块。在 body 部分一般就是具体的程序代码了，包括类和方法。也有一些测试库会加上“if __name__ == '__main__': ”在最后，用来判断是否直接运行的这个 py 文件。

10.2.3　测试库命名

在 Python 里来说，每一个测试库的目录名或文件名都是它的 module，通常在 module 里会有一个 class 类，一般我们都要求 class 的名字和 module 的名字是相同的。比如 ExampleLibrary.py 的文件名是 ExampleLibrary，那么代码里的 class ExampleLibrary 就是相同的，这样在测试案例中加载测试库就可以直接使用 ExampleLibrary 来加载。所有在这个 class 里的方法都可以在测试案例中使用，不过要除去私有方法。ExampleLibrary 里有两个方法，gen_nums 和_gen_nums，通常带有“_”开头的方法都算做私有方法，在 Robot Framework 里会自动过滤掉“_”开头的方法，所以当你在 RIDE 里按 F5 快捷键，查看 ExampleLibrary 的关键字列表里只会看到 gen_nums。

细心的朋友会看到这里还有一个 class OtherLibrary，前面提到了一般要求 class 的名字和 module 的名字是相同的，但是其实也允许它不同，那这个 OtherLibrary 就是不同的了。如果想要在 Robot Framework 里使用这个类里面的方法也是可以的，一般有两种手段，一个是在 ExampleLibrary 里去把 OtherLibrary 的方法引用过来，实际上相当于套了个壳儿，在加载的时候还是加载的 ExampleLibrary 的方法，但是底层是调用 OtherLibrary 的方法；另一个是在加载 Library 的时候用 module.class 来加载，例如可以用 ExampleLibrary.OtherLibrary 来进行加载。所以大家在对测试库进行命名的时候需要注意这一点。

10.2.4　测试库头部

说完命名，来看一下测试库的“头部”，首先是编码声明，#encoding=utf-8 是用来声明当前文件是 UTF-8 编码的，增加这个声明的好处是，你的关键字注释可以用中文了。接着是一些必要的 import，在测试库的代码里可能会用到一些不同模块的方法，甚至于是其他测试库的接口，而这些方法的使用一般都需要我们先 import。

```
import string
import random
from robot.api import logger
import logging
```

比如这里的 string、random 都是 python 的一些基础库，而 robot.api 则是 Robot Framework 提供的接口，这里我们用它提供的 logger 对象。

10.2.5 测试库配置项

测试库有一些基础配置项，放到 class 里，这里介绍几个主要的。

```
ROBOT_LIBRARY_SCOPE = 'GLOBAL'
```

scope 可以理解为范围或者作用域，可设置的值有 TEST CASE,TEST SUITE,GLOBAL。如果设置为 TEST CASE，那么每个 Case 都会创建一个新的该测试库的实例。换个容易理解的说法，比如有个计数关键字，每次调用这个关键字就“+1”。如果设置为 TEST CASE，在一个 Case 里会一直加下去，但是在另一个 Case 里再调用计数关键字就会创建一个新的实例，重头开始计数了。如果设置成 TEST SUITE，那么在当前的测试套件里就会一直计数，而新的测试套件就会创建一个新的实例，重头计数。如果设置成 GLOBAL，那么就只有一个实例，只要不清零，计数会一直进行下去。

```
ROBOT_LIBRARY_VERSION = 0.1
```

测试库的版本号，如果不设置的话会尝试读取“__version__”属性。

```
ROBOT_LIBRARY_DOC_FORMAT = 'reST'
```

测试库的文档格式，支持 ROBOT/HTML/TEXT/reST，其中 reST 这种格式如果你要用 libdoc 生成文档时，需要先安装 docutils。

10.2.6 测试库文档

测试库文档其实就是代码里的 document 注释，通常用 3 个双引号括起来：

```
"""Library 文档 *斜体* 这个文档用的是 reST 结构 reStructuredText__.

    __ http://qitaos.github.io
"""
```

无论是 Library 的还是关键字的 document，通常都是建议写在代码注释里，这样有一个好处，Robot Framework 提供了一个 libdoc 的工具，可以很方便地生成一份测试库文档，

具体命令行参考：

```
   python -m robot.libdoc -f html ExampleLibrary.py ExampleLibrary.html
   python -m robot.libdoc -f html ExampleLibrary.OtherLibrary
ExampleLibrary.OtherLibrary.html
```

它会将测试库里的 document 注释都提取出来，然后结合关键字一起生成这样一份文档。很多测试库都是这样生成文档的，不需要去手动再维护一份文档，每次代码有了更新就同步更新 document 注释，然后重新生成一份新的文档即可。这两份文档都在 github 上存放。

10.2.7　测试库关键字命名

对于测试库里的关键字方法的命名是用来给测试案例使用的，除了前面说到的“_”开头的关键字方法不会看到，其他的关键字都可以直接被使用，并且在使用时忽略大小写、空格或下划线的。比如 count，你可以在案例里写 Count、count，甚至于“c o u n t”都可以调用到底层的 count 方法，比如 gen_nums，在案例里写 gen_nums 或者 gen nums 也一样都可以调用到底层的 gen_nums 方法。

还有一些自定义方法名或者是通过修饰符自定义方法名的处理，建议参考一下官方文档。

10.2.8　测试库关键字参数

在 class 里的关键字方法的参数，第一个参数总是 self，之后才是函数真正需要的参数。在用户关键字里我们知道，可以给每个参数添加默认值。在测试库里的关键字参数也是一样可以添加默认值的，规则和用户关键字里的一样，把必填参数放在前面，有默认值的参数放在后面，List 参数放在最后。

参数的种类也可以按照之前讲的来理解，不带“*”的就是 Scala 型的参数，带“*”的就是 List 型参数。例如：

```
   def arg_demo(self, arg1, arg2= 2, *args):
```

前面两种都好理解，除此之外还有带“**”的参数，例如：

```
   def freearg_demo(self, **freearg):
```

这个其实是相当于 Dictionary 型的参数，这种参数传值的时候是要传 key=value 这样的值的。

最后补充一点，如同之前介绍过 Robot Framework 的变量默认是字符型的，在底层 Library 里所有的参数类型默认也都是 string，如果想要特定的格式，比如 int，则需要在关键字里进行转换。

10.2.9 测试库的参数

测试库本身也是可以支持参数的，并可以在 import 的时候提供参数输入。一般测试库的参数都是在测试库的初始化方法“__init__”里添加，为了避免使用测试库的人不知道要添加参数，通常建议这些参数最好是有默认值的。例如：

```
def __init__(self, arg = 1):
```

10.2.10 测试库关键字返回值

测试库里的关键字返回也是和之前介绍过的用户关键字返回一样，可以返回 scala 值，也可以返回 List 类型的值，当然还可以返回一个 object 对象，如果是 object 对象，那么在测试案例中就要用${object.attribute}来使用，像前面做接口测试的例子里就有这样的。

10.2.11 测试库日志打印

在测试库的关键字里非常有必要增加日志打印，一方面可以告诉用户你的一些操作，另一方面也可以把一些异常信息通过日志告诉用户，比如参数错误之类的。

（1）“print”最简单的是直接用 print，例如：

```
print '*INFO* Get random number string: %s' % s
```

其中“*INFO*”是表示 log 的 level，主要的 level 有 TRACE, DEBUG, INFO, WARN, ERROR 和 HTML。

（2）“log to console”主要用来打印日志到控制台，也可以用“sys.__stdout__”或者“sys.__stderr__”来进行打印。例如：

```
sys.__stdout__.write('Got arg %s\n' % arg)
```

（3）“logging API”Robot Framework 本身提供了对外的接口 robot.api，里面就有 logger 这个对象，使用之前要先 import，然后就可以用 logger 的打印方法了，比如：

```
from robot.api import logger
logger.debug('self._counter += 1')
logger.info('self._counter += 1')
```

（4）“python logging”Python 自带了基础的 logging 模块也可以用于日志打印，同样要先 import 再使用，用法和 logger 类似：

```
import logging
logging.info('self._counter = 0')
```

10.2.12　对测试库做测试

一般测试库里都会有自己的测试，因为测试库本身也是程序，那么也要验证功能的正确性。通常可以用单元测试或者验收测试来做，这里笔者会加一些验收测试的案例。

针对 ExampleLibrary 的测试案例，见表 10-2-1。

表 10-2-1

Set Log Level	DEBUG		
${num}	Set Variable	4	
${s}	Gen Nums	${num}	
${len}	Get Length	${s}	
Should Be Equal As Integers	${num}	${len}	
@{list}	Create List	4	5
Arg Demo	1	3	@{list}
Freearg Demo	test=1	aaa=b	

针对 ExampleLibrary.OtherLibrary 的测试案例，见表 10-2-2。

表 10-2-2

${count}	Count	
Should Be Equal As Integers	${count}	1
${count}	Count	
Should Be Equal As Integers	${count}	2
Clear Counter		
${count}	Count	
Should Be Equal As Integers	${count}	1

10.2.13　发布测试库

通常测试库的源码都是保存在 github 上面的。如果你是自己用，发布与否其实都无所

谓了。如果想提供给其他人使用，那就要考虑一下使用者了，最好是能做好安装文件。如果想放到 Python 官网也可以，那要按照 Python 官方要求去进行申请。放到 Python 官网的好处是可以用 pip 安装，如果只是放 github 上，一般都是下载后用“python setup.py install”来安装。

这里主要介绍普通方式安装的，这种比较推荐 Setuptools 安装框架。对于发布的测试库来说，一般目录结构如下（以 QTLibrary 为例）：

```
-src
    QTLibrary
-doc
-test
README.md
setup.py
```

其中重要的就是 setup.py 这个文件了，其中涉及安装部分的一些配置如下：

```
setup(name      = 'robotframework-qtlibrary',
      version    = VERSION,
      description = 'QTLibrary for Robot Framework',
      long_description = DESCRIPTION,
      author     = 'Qitao',
      author_email = 'qitaos@gmail.com',
      url        = 'https://github.com/qitaos/Robotframework-QTLibrary',
      license    = 'No License',
      platforms  = 'any',
      classifiers  = [
                  ],
      install_requires = [

                      ],
      py_modules=['ez_setup'],
      package_dir= {'' : 'src'},
      packages   = ['QTLibrary','QTLibrary.keywords'],
      include_package_data = True,
      )
```

前面都是一些版权说明，比较核心的部分就是 install_requires，如果你的安装包依赖其他的库，则需要列在这里。package_dir 是你的包目录，一般都是在 src 源码包。packages 一般是你的测试库的 class，把需要的包列在这里。还有一些参数配置就需要查阅一下 Setuptools 的具体说明了。

QTLibrary 的这个测试库的参考地址：

https://github.com/qitaos/robotframework-QTLibrary

10.3　小结

本章内容并不太多，实际上还有一些深入的内容并没有介绍，只能算是个引子，引导大家如何上手做一个测试库，至少是一个能用的测试库。有部分测试库要注意的要点参考了官方文档，因为怕遗漏了什么重要的东西。当然有些内容还用不上的，或者对入门来说可以先不考虑的，这里就没有加。所以如果读者想详细完整地了解做一个测试库的话，推荐看看官方文档，参考地址如下：

http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

在测试库“这个地方”来说，其实更多的还是看大家各自的需求，说到底就不是测试了，而是开发。从开发实现一个功能来说，有很多种设计方案，没有哪个是最好的，只有合适还是不合适。所以从这个角度来说，笔者的例子也做的比较简单，只希望大家通过例子了解该如何去做，而不是让例子限制大家的思维。希望有志于开发或维护测试库的朋友，可以发挥自己的编程能力，也欢迎大家一起探讨研究。

结　　语

其实去年就该写这本书了，一直没有静下心来写，直到今年年初才痛下决心要把这本书写出来。从一月份到现在，原本和出版社约的是五月份交稿，实际上到了五月份才完成了 60%左右，然后拖到了八月份，终于完稿了。中间也曾经想过要放弃，也曾经因为工作的不顺心耽误了写作，甚至因为自己的“拖延症”强迫用番茄工作法来督促自己，当然能够坚持完成这本书，最主要还是我太太持续对我的鼓励和“逼迫”，同时也和 QQ 以及微信上的朋友们的支持是分不开的。

我一直在想，为什么要写这本书，要赚钱么？Robot Framework 在国内来讲其实还是有些小众了，所以指望这本书大卖赚钱是不可能的，我甚至还想着如果卖不掉，是不是自己直接找出版社采购一批放到淘宝或者微店上去卖。要名气么？确实有可能因为一本书提高一些名气，不过单纯想靠一本书捧红自己还是有点难，现在写书的人太多了。有人说你不为名不为利，写它干什么呢？其实关于 Robot Framework 在国内来说，我不是第一个“尝鲜”的人，有很多走在我前面的人或者一起同行的人，这几年但凡是我去参加什么活动，基本上主题都离不开 Robot Framework。从吴穹博士把它推荐给我，从我学习了解 Robot Framework 开始，我就被这个框架吸引了。以前我是用 QTP 的，而看到这个框架，为它的轻量赞叹，为它的支持广泛和良好的可拓展性赞叹。所以在我学习的同时，就开始在 CSDN 的博客上写了介绍，并随着深入的学习，我重新梳理了知识点，并在博客上专题连载。然后加入了吴博士的分层自动化测试群，认识了很多朋友，然后加入了恒总的 Robot Framework 群，发现也有很多人在使用 Robot Framework，并且很多朋友都是看了我的博客一起学习的，很开心能有这么多小伙伴一路同行。从 2011 年接触 Robot Framework 到现在，也有将近 4 年的时间了，曾经也整天泡在 QQ 群里解答网友的问题，然后发现国内了解并熟悉 Robot Framework 的人越来越多了，但是一直没有太多的中文说明，官方的文档都是英文的，很多人不习惯看英文文档，其实我英文也不太好，但是我学习的过程，大部分也都是看官方英文文档的，遇到不会的就在网上翻译一下，加上练习，才让我在 Robot

Framework 领域走得稍微快一些。当然，我对于 Robot Framework 也不敢说完全深入了解，也存在一些知识死角。所以我和吴穹博士沟通了一下，他也建议我出这样的一本书，所以就写了。我写这本书的目的其实是希望降低新入门的同学的门槛，同时希望这本书能成为国内第一本全面介绍 Robot Framework 的中文书籍，我希望能有更多的人掌握它之后，一起深入研究，共同交流共同进步，在测试行业日趋低迷的态势下，能够尽力提升测试人员的技能水平，提高个人的竞争力，为测试这个行业贡献一些绵薄之力。目标有些远大，不敢说一定能怎么样，只能说先从这本书开始努力吧！

这本书其实有部分内容也在博客上有，只不过重新进行了梳理，希望能系统化地介绍 Robot Framework，让学习的朋友可以循序渐进地学习了解，深入的部分介绍的不是太多，因为深入的内容更需要大家自己发挥，总体说来这本书更适合不了解 Robot Framework 的、刚刚开始接触的朋友，也希望对已经在使用的同学有所帮助。

最后再次感谢吴穹博士，感谢 monkey 陈晔和徐毅老师百忙之中帮我审稿，感谢芈峮、刘晓光、恒总、葡萄、AntZ、喜洋洋等 QQ 群友们，感谢 Robot Framework 公众号的订阅者们，感谢所有一直在支持我的朋友们，谢谢大家！

2015 年 8 月 9 日晚 于深圳家中完稿。

参考资料

http://robotframework.org/

http://appium.io/